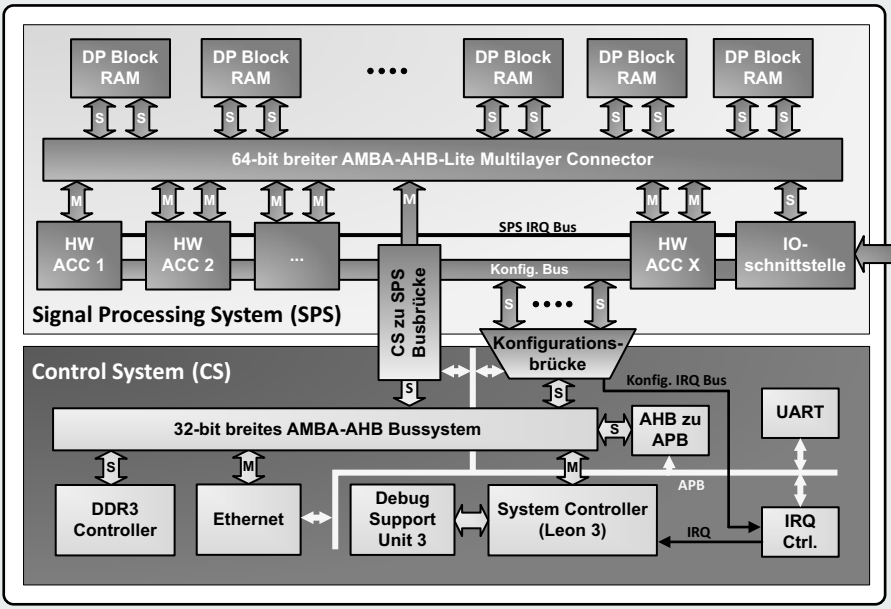


Joachim Meyer

HW/SW Co-Design Framework für Hochgeschwindigkeits-OFDM Signalverarbeitung

HWSW Co-Design Framework für Hochgeschwindigkeits-OFDM Signalverarbeitung

Zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEURS

von der Fakultät für
Elektrotechnik und Informationstechnik
des Karlsruher Institut für Technologie (KIT)
genehmigte

DISSERTATION

von

Dipl.-Ing. Joachim Heribert Meyer

geb. in Bühl (Baden)

Tag der mündlichen Prüfung:
13.11.2014

Hauptreferent: Prof. Dr.-Ing. Dr. h. c. Jürgen Becker
Korreferent: Prof. Dr.-Ing. Michael Hübner

HW/SW Co-Design Framework für
Hochgeschwindigkeits-OFDM Signalverarbeitung

1. Auflage: Februar 2015

Für meine Eltern

Manfred und Chrysantha Meyer

Zusammenfassung

Im Mittelpunkt dieser Arbeit steht ein Framework, welches es auch Anwendern ohne Kenntnisse des Schaltungsentwurfs ermöglicht, OFDM-Systeme der verschiedensten Konfigurationen und Anforderungen zu erstellen. Diese sollen einen bisher nicht möglichen Kompromiss zwischen Leistungsfähigkeit und Flexibilität bieten. Einerseits werden Datenraten im Gigabit-Bereich angestrebt, andererseits soll ein Abändern der wichtigsten OFDM-Parameter durch Software jederzeit möglich sein.

Zunächst werden die Grundlagen über OFDM-Kommunikation, FPGAs und Signalverarbeitungsarchitekturen eingeführt, bevor das Folgekapitel auf den aktuellen Forschungsstand eingeht. Es zeigt sich, dass im Rahmen von Hochgeschwindigkeits-OFDM derzeit durchgehend mittels FPGA realisierte, unflexible, dedizierte Schaltungen zum Einsatz kommen und dass die Datenraten flexibler Ansätze bisher wesentlich geringer sind als in dieser Arbeit gefordert.

Ausgehend von einem Anwendungsszenario aus dem Radio-Backhauling Bereich wird die Signalverarbeitung eines typischen OFDM-Systems als Basis eingeführt. Nach einer Zerlegung dieser Signalverarbeitungskette mittels *Top-Down Ansatz* folgt die Bestimmung derjenigen Signalverarbeitungsschritte, welche im Sinne eines *Hardware/Software Co-Designs* zur Beschleunigung durch Hardware besonders geeignet sind. Schließlich werden fundamentale Systemparameter der Architektur diskutiert und festgelegt, bevor auf den Entwurf und die Implementierung der einzelnen Hardwarebeschleuniger eingegangen wird.

Die Ergebnisse der Experimente aus Kapitel 5 demonstrieren sowohl die Funktionalität als auch die Qualität, welche sich mit den Prototypen erzielen lässt, die aus den entwickelten Komponenten erstellt wurden. Sie bilden die *schnellsten veröffentlichten Prototypen programmierbarer Hochgeschwindigkeits-OFDM Signalverarbeitungssysteme*, deren Parameter sich flexibel anpassen lassen und bei denen die Signalverarbeitung in Echtzeit durchgeführt wird. Die 8-Stream Sender Variante erzielte mittels einer 64-QAM Modulation der Träger eine Datenrate von *über 10 Gb/s*. Aufgrund des erhöhten Ressourcenverbrauchs war mit dem verfügbaren FPGA lediglich eine 4-Stream Variante des Empfängers realisierbar, wodurch die in Experimenten erreichte maximale Datenrate bei *über 6 Gb/s* liegt. Die Signalqualität genügte bei allen Messungen den Anforderungen, welche von modernen Fehlerkorrekturverfahren für den Aufbau fehlerfreier Systeme gestellt werden.

Zur vollständigen Erstellung eines Systems, das dem Anwendungsszenario genügt, wurde im Rahmen dieser Arbeit zusätzlich ein neuartiger Digital Down Converter Prototyp entwickelt, der eine Vorverarbeitung von Hochgeschwindigkeits-OFDM Signalen auf Empfängerseite erlaubt. Nach einer detaillierten Beschreibung der Optimierungen, welche notwendig sind um die anvisierte Zielfrequenz für das Taktsignal zu erreichen, werden der aufgebaute Prototyp sowie die durchgeführten Experimente beschrieben. Mit einem Durchsatz von bis zu 25 GSa/s handelt es sich bei diesem Prototypen um den *breitbandigsten Digital Down Converter*, der bisher veröffentlicht wurde.

Abschließend folgt die Beschreibung der Integration der entwickelten Komponenten in das Framework. Durch die erstellte *grafische Benutzeroberfläche* wird in Kombination mit den FPGA-Entwicklungswerkzeugen ein komfortabler Entwurfsfluss zur Erstellung einer Vielzahl von Systemen erreicht. Hierdurch ist es auch Fachfremden möglich, auf einfachste Weise komplexe, angepasste Systeme zu erstellen und mit diesen eine OFDM-Entwurfsraumexploration durchzuführen. Die Programmierung der erzeugten Systeme wird durch eine große Anzahl an vorgefertigten Treiber- und Testfunktionen unterstützt. Die mögliche Vielfalt an Systemvarianten, welche sich mit Hilfe des Frameworks erstellen lässt, reicht von einem einfachen Prozessor-System mit Hardwarebeschleuniger-Unterstützung einzelner Algorithmen für Datenraten im Megabit Bereich bis hin zu heterogenen MPSoCs mit Datenraten im Gigabit-Bereich.

Abstract

The focus of this thesis is to create a framework that even allows users without knowledge of circuit design to set up OFDM systems of various configurations and requirements. These systems shall offer an unprecedented compromise between performance and flexibility. On the one hand, the system targets data rates in the gigabit range; on the other hand it provides the possibility of adjusting the most important OFDM parameters by means of software. First the fundamentals of OFDM communication, FPGAs and signal processing architectures are introduced before the next section depicts the state of the art. It is found that in the context of high-speed OFDM inflexible, dedicated circuits realized by FPGA dominate the scene and that the data rates achieved by more flexible approaches so far are much lower than those this work aims for. On the basis of an application scenario from the radio backhauling area the signal processing of a typical OFDM system is introduced. After the decomposition of the signal processing chain by means of a *top-down* approach those signal processing steps are identified, which qualify for hardware acceleration in sense of a *hardware/software co-design*. The definition of fundamental parameters of the system architecture follow before the design and implementation of the individual hardware accelerators are discussed. The results of the experiments in chapter 5 demonstrate both the functionality as well as the quality which can be achieved with prototypes composed by developed components. They represent the *fastest published prototypes of software-defined high-speed OFDM signal processing systems*, whose parameters can be adapted and which feature real-time signal processing. The 8-stream version of the transmitter achieved a data rate of *more than 10 Gb/s* using 64-QAM modulations for the subcarriers. Because of the increased resource consumption of the receiver, the available FPGA only allowed a 4-stream version, thereby the maximum data rate of the receiver experiments was *more than 6 Gb/s*. For all measurements the signal quality was sufficient to fulfil the requirements for building up error free systems using modern error correction methods. In order to complete the creation of a system that satisfies application scenario requirements, a prototype of a novel high-speed digital down-converter was developed additionally which enables the pre-processing of the OFDM signals on the receiver side. After a detailed description of the optimizations which are necessary to achieve timing closure with the intended clock frequency, the prototype set-up and the experiments carried out are specified. With a throughput of up to 25 GSa/s this prototype is the *fastest*

digital down-converter which has been published. Finally, the developed components are integrated into a framework. A comfortable tool flow for the creation of a wide variety of systems is provided by the combination of the developed *graphical user interface* and the design tools of the FPGA manufacturer. This also enables non-specialists to easily create complex, customized systems and use them in order to perform an OFDM design space exploration. Writing software for those systems is supported by a large number of driver and testing functions ready to use. The diversity of system variants, which can be created with the help of the framework, covers single processor systems with hardware acceleration support for individual algorithms going for data rates in the megabit range as well as heterogeneous MPSoCs with data rates in the gigabit range.

Vorwort

Die vorliegende Arbeit entstand während meiner Zeit als wissenschaftlicher Mitarbeiter am Institut für Technik der Informationsverarbeitung (ITIV) des Karlsruher Institut für Technologie (KIT). Es war eine wundervolle Zeit, in welcher ich unzählige Möglichkeiten bekam, zu lernen, zu lachen, Erfahrungen in vielerlei Hinsicht zu sammeln, viele tolle Menschen kennen zu lernen, ein Stück von der Welt zu sehen und mich selbst zu verwirklichen. Ich schätze mich glücklich, dass ich die Gelegenheit hatte, hier zu promovieren und daher möchte ich mich hiermit recht herzlich bei allen bedanken. Leider reicht der Platz nicht aus, alle Menschen namentlich zu nennen, welche in den letzten Jahren meine Arbeit und meinen Alltag bereichert haben. Ich hoffe sie verzeihen mir, dass ich hier nur einige erwähne.

Zuerst möchte ich mich natürlich ganz besonders bei meinem Doktorvater Prof. Jürgen Becker für die exzellente Betreuung und das enorme Vertrauen in mich und meine Arbeit bedanken. Die Freiheiten, welche er mir in den letzten Jahren ließ, sorgten dafür, dass falls einmal nötig, ich auch am Wochenende gerne zur Arbeit kam. Gleiches gilt für Prof. Michael Hübner von der Ruhr-Universität Bochum. Ein herzliches „Danke“, nicht nur für die Übernahme des Koreferats, auch für seine tolle Unterstützung während seiner Zeit am KIT.

Den Professoren Jürgen Leuthold, Wolfgang Freude und Christian Koos vom IPQ möchte ich ebenfalls meinen Dank für die gute Zusammenarbeit und ihre Geduld aussprechen. Während den wöchentlichen Treffen ermöglichten sie mir tiefe Einblicke in die Welt der optischen Kommunikationssysteme.

Ebenso danken will ich meinen Kollegen und Studenten am ITIV, die meine Zeit als Doktorand so besonders machten. Allen voran stand mir Michael Dreschmann als treuer Freund und Bürokollege in allen Situationen hilfreich zur Seite. Ohne dich wäre das ITIV nicht das ITIV. Auch Oliver Sander, Michael Rückauer, Falco Bapp und all die anderen Kollegen des ITIV haben mit ihrer herausragenden Unterstützung, den vielen interessanten Gesprächen und mit ihrem tollen Humor zum Erfolg dieser Arbeit beigetragen.

Ebenso dankbar bin ich René Schmogrow, Phillip Schindler und Djorn Karnik sowie den anderen Mitarbeitern des IPQ für ihre Unterstützung im Bereich der Optik und Hochfrequenztechnik, besonders in Bezug auf die durchgeführten Experimente.

Auch meinen Freunden, welche mir immer einen Ausgleich zum Institutsalltag ermöglichten und daher auch indirekt zu dieser Arbeit beigetragen haben möchte ich danken, im speziellen Mike und Boris aus Ottersweier, Artur, Mark und Nadine aus Karlsruhe, Jeremy, sowie Marc und Vanessa aus der WG.

Ein ganz spezieller Dank gilt meiner Familie, ganz besonders meinen Eltern als auch Anne, Imme, Joni und Leo. Ihnen verdanke ich alles.

Schließlich möchte ich allen danken, die durch ihre Anregungen, Korrekturen und Diskussionen auf die eine oder andere Weise zur Erstellung dieses Manuskripts beigetragen haben. Besonders erwähnen möchte ich meine Freundin Dani sowie Judith Marnet.

Karlsruhe, im Februar 2015
Joachim Meyer

Inhaltsverzeichnis

1. Einleitung

Eine Schlüsseltechnologie für die Entwicklung einer modernen Gesellschaft stellt eine ausgeprägte Informations- und Kommunikationstechnik dar. Sie bildet nicht nur eine Grundlage um Wissen effizient zu sammeln und vielen Menschen auf einfache Weise zugänglich zu machen, sie ist auch aus dem modernen Leben eines jeden nicht mehr wegzudenken.

Man beginnt den Tag mit dem Alarm eines Weckers, welcher seine Uhrzeit über Funk (DCF77) synchronisiert, schaut am Frühstückstisch die Tagesthemen per Internet auf dem Smartphone und umfährt den Stau auf dem Weg zur Arbeit mit Hilfe des Navigationssystems. Auf der Arbeit verbringt man dann den Großteil des Tages am Computer oder am Telefon und am Abend lädt man sich den neusten Blockbuster per Video-On-Demand direkt ins Wohnzimmer.

Immer mehr Alltagsgegenstände wie der Kühlschrank, der Autoschlüssel oder auch die Haustür werden im Sinne des Ubiquitous Computing (vgl. [93]) durch Computer gesteuert und mit dem Internet vernetzt, um entweder aus der Ferne kontrolliert zu werden oder um sich autark aktuelle Informationen zu beschaffen und sich dadurch den gegebenen Umständen besser anzupassen.

Gerade die Weiterentwicklung der drahtlosen Kommunikation spielt hierbei eine große Rolle. Während vor zehn Jahren meist nur technisch affine Menschen ein lokales Netzwerk sowie einen Hochgeschwindigkeits-Internetanschluss besaßen, fällt es heutzutage schwer, auch nur einen Fernseher zu kaufen, der sich nicht auf Knopfdruck ins eigene Drahtlosnetzwerk einwählt, um die neuesten Nachrichten der bevorzugten sozialen Plattform anzuzeigen.

Die Anforderungen an die Technik sind hierbei insgesamt sehr unterschiedlich, siehe Abbildung 1.1. So reichen die benötigten Datenraten von einigen wenigen Kilobits pro Sekunde, wie zum Beispiel im Bereich der Nahfeldkommunikation (Near Field Communication, NFC) über Hunderte von Megabit pro Sekunde beim neuen Mobilfunkstandard Long Term Evolution (LTE) bis zu einem Gigabit pro Sekunde für den zukünftigen WLAN Standard (IEEE 802.11ac).

Schaut man auf die Entwicklung des Internets und den darüber stattfindenden, weltweiten Datenverkehr, zeichnet sich ein enormer jährlicher Anstieg ab, eine Sättigung ist nicht in Sicht, vergleiche Abbildung 1.2, vgl. [20]. Die Entwicklungen moderner Kommunikationssysteme in Richtung immer größerer Datenraten

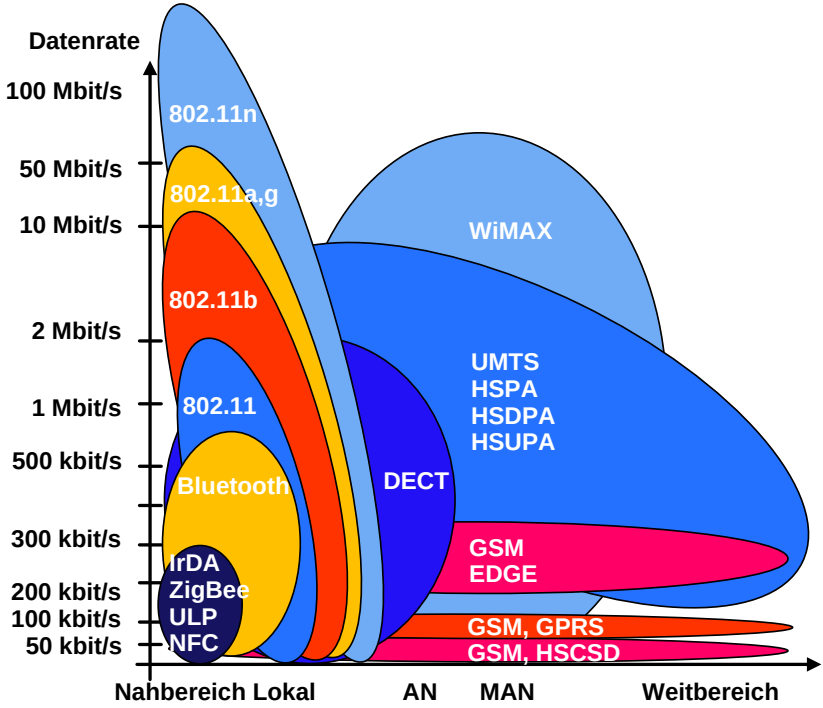


Abbildung 1.1.: Einteilung einiger aktueller Funkstandards in Bezug auf ihre Datenrate und ihre Reichweite [42].

sorgt letztendlich auch für immer höhere Anforderungen an die Leistungsfähigkeit der Signalverarbeitung.

Im Bereich der drahtlosen Kommunikation hat sich gerade für hohe Datenraten, in vielen Bereichen das *Orthogonale Frequenzmultiplexverfahren* (Orthogonal Frequency-Division Multiplexing, OFDM) durchgesetzt. Es bietet als Mehrträgerübertragungsverfahren Vorteile bei der Behandlung von Mehrwegeausbreitung, erreicht dennoch eine ähnliche Bandbreiteneffizienz wie entsprechende Einträgerverfahren. So verwendet beispielsweise der aktuelle Mobilfunkstandard LTE das *Orthogonal Frequency-Division Multiple Access* (OFDMA) Verfahren im Downstream, also von Mobilfunkstation zu Smartphone.

Auch bei Radar- Anwendungen stoßen OFDM Systeme auf breites Interesse, vergleiche [87] und [49]. Hier steht zwar nicht die maximale Datenrate im Vorder-

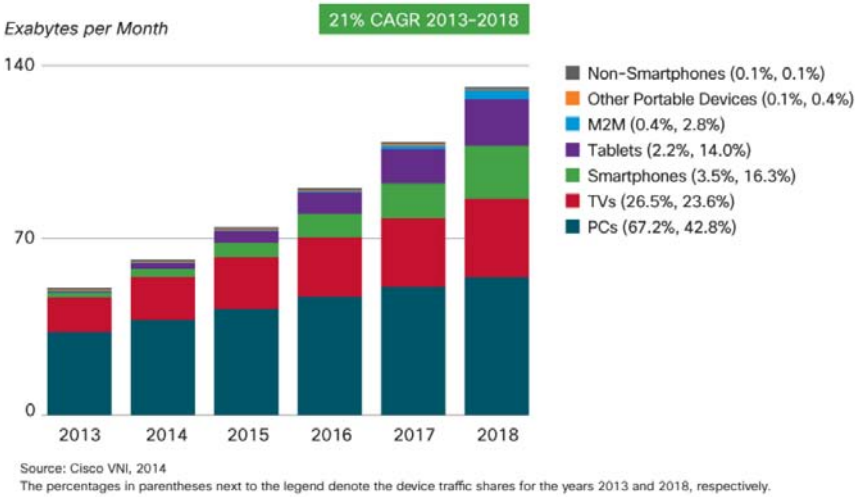


Abbildung 1.2.: Globaler Internet Protocol (IP) Verkehr aufgeschlüsselt nach Gerätetyp [20].

grund, jedoch stellen Techniken wie *Multiple Input Multiple Output* (MIMO) oder die *Ultra-Breitband-Technologie* (Ultra-wideband, UWB) auch hier sehr hohe Anforderungen an die digitale Signalverarbeitung.

Dieses steigende Interesse an Orthogonal Frequency-Division Multiplexing sowie an OFDM ähnlichen Techniken wie *Single-carrier FDM* (SC-FDM), als auch der Fortschritt der technischen Möglichkeiten im Bereich der Signalverarbeitung und Analogwandler führte dazu, dass sich heute auch im Bereich der optischen Kommunikationstechnik, speziell in der Forschung und Entwicklung, viele Gruppen mit OFDM beschäftigen. Gerade durch die angesprochene Bandbreiteneffizienz wird OFDM seit einigen Jahren im Bereich von Zugangsnetzwerken untersucht, siehe [48] und [4]. Obwohl auch hier, ähnlich zur drahtlosen Übertragung, eine effiziente und kostengünstige Realisierung wichtig ist, besteht gerade im Bereich der optischen Übertragung ein Forschungsschwerpunkt darin, die Datenraten an ihr Maximum zu treiben.

1.1. Motivation

Da die immer weiter steigenden Anforderungen an OFDM Systeme in einer Vielzahl von Anwendungsfällen die Leistungsfähigkeit einfacher Rechensysteme überschreiten, sind viele Forschungsgruppen aus den unterschiedlichsten Bereichen gezwungen, eine adäquate Lösung zu finden. Soweit möglich, führen viele ihre Experimente daher nur noch offline durch. Sendedaten werden dabei bereits zuvor berechnet, das Ergebnis muss während des Experiments dann lediglich noch *abgespielt* werden, beispielsweise durch einen *Arbiträrgenerator* (Arbitrary Waveform Generator, AWG). Für den Empfang wird das Eingangssignal eine gewisse Zeit aufgezeichnet, um dann anschließend ausgewertet zu werden, wofür jedoch beliebig viel Zeit zur Verfügung steht.

Andere Forschungsgruppen gehen noch weiter und beschränken ihre Arbeit vollständig auf Simulationen. Dies ist zwar in vielen Fällen legitim, allerdings sind Experimente mit Echtzeitsignalverarbeitung in der Regel notwendig, um wirklich alle Effekte eines Systems erfassen und dadurch korrekte Schlussfolgerungen ziehen zu können. Daher suchen sich viele Forschungsgruppen neue Möglichkeiten, eine entsprechende Datenverarbeitung trotz der hohen Anforderungen umzusetzen.

Eine dieser Möglichkeiten stellen *Field Programmable Gate Arrays* (FPGA) dar. Durch die Kombination von enormer Flexibilität, sehr hoher Ein- und Ausgangsbandbreite sowie höchster Rechenkapazität eignen sie sich besonders zur Realisierung von Algorithmen der Signalverarbeitung. In nahezu allen Bereichen, ob beim Radar oder in der optischen Datenübertragung, zur Signalverarbeitung in OFDM basierten Übertragungssystemen lassen sich gerade bei sehr hohen Datenraten immer häufiger Realisierungen auf FPGA Basis finden.

Ein Problem, welches sich jedoch vielen Forschungsgruppen bei der Verwendung von FPGAs stellt, besteht darin, dass FPGAs sich traditionell nicht wie andere Bausteine mit einer der weit verbreiteten Hochsprachen wie C++ oder Java programmieren lassen. Vielmehr wird ein Hardwareentwurf durchgeführt, die Bestandteile eines Systems werden Stück für Stück mittels speziellen *Hardwarebeschreibungssprachen* (Hardware Description Language, HDL) wie VHDL oder Verilog beschrieben und zu einer komplexen Schaltung zusammengesetzt. Dadurch lässt sich FPGA-Technologie in der Regel nur von Anwendern nutzen, die zum einen über fundiertes Wissen im Bereich des digitalen Hardwareentwurfes verfügen und zum anderen zumindest ein Basiswissen speziell über FPGAs besitzen.

Daher besteht ein großes Bemühen, sowohl von Seiten der Hersteller als auch von Seiten der Forschung, das Programmieren von FPGAs und somit das Erstellen von Hardwareschaltungen zu erleichtern. So wird zum Beispiel angestrebt, durch Werkzeuge wie *LabVIEW* und entsprechende Erweiterungen (beispielswei-

se NI LabVIEW FPGA Module [67]) eine Art Baukastensystem für verschiedene vorgefertigte Module anzubieten, in diesem Falle mit Fokus auf Signalverarbeitung. Ähnliches existiert mit dem *Xilinx Platform Studio* (XPS) von Xilinx auch für Prozessorsysteme. Für die existierenden Module funktioniert dies in weiten Bereichen ganz gut. Dabei ist man allerdings auf die vorgefertigten Bibliotheken beschränkt, welche wiederum selten hochspezialisierte Anwendungsfälle wie *Hochgeschwindigkeits-OFDM* abdecken.

Ein anderes Konzept sieht die HDL Generierung aus Hochsprachen vor. So erlaubt das neue *Vivado High-Level Synthesis* Werkzeug von Xilinx eine Beschreibung von Hardware mit Hilfe von C, C++ oder SystemC, der VHDL-Coder von Mathworks erstellt sogar aus Matlab-Code entsprechende Hardware. Zum einen funktioniert dies allerdings nicht mit einem beliebigen Programmcode, es müssen wieder bestimmte Regeln in der Programmierung eingehalten werden, da man auch hier Hardwareschaltungen beschreibt. Zum anderen erreicht man in der Regel nicht die Qualität welche ein erfahrener Designer mittels Beschreibung auf Registertransferebene erzielt, vgl. [95] und [85].

Des Weiteren lassen sich OFDM-Parameter von Systemen, in welchen die Algorithmen direkt als Hardwareschaltung umgesetzt und in einem FPGA implementiert wurden, oft nur schwer oder gar nicht ändern. So können gewollte oder notwendige Parameteränderungen einen vollständigen Neuentwurf der Hardwarebeschreibung bedeuten. Gerade Techniken wie OFDM besitzen jedoch eine Vielzahl von Parametern, deren Einfluss auf ein System essentiell sein kann. Tabelle 1.1 zeigt die Vielzahl an Konfigurationen anhand einer kleinen Auswahl an OFDM-Parametern und Systemen und verdeutlicht damit den großen Mehrwert, welchen Systeme bieten die flexibel konfigurierbar sind.

Ziel dieser Arbeit ist es, ein Framework zu erstellen, welches es mit Hilfe einer Sammlung von Hardware und Software ermöglicht, auf einfachste Weise und auch für Anwender ohne besondere Vorkenntnisse die Signalverarbeitung für ein weites Spektrum an OFDM-Kommunikationssystemen aufzubauen. Dabei soll es nicht nur möglich sein, außergewöhnlich hohe Datenraten zu erreichen, sondern auch das Anpassen der OFDM-Parameter zur Laufzeit soll sehr einfach erfolgen können, idealerweise durch das Abändern der Software. Die Programmierung sollte durch eine gängige Hochsprache stattfinden und durch das Framework so unterstützt werden, dass auch hierfür keine speziellen Kenntnisse erforderlich sind.

So sollen auch Anwender, welche lediglich rudimentäre Programmiererfahrung besitzen, in der Lage sein, angepasste OFDM-Signalverarbeitungssysteme zu erstellen und mit deren Hilfe ihre neuen Ideen und Anwendungen im Labor zu testen, zu analysieren oder zuvor simulierte Ergebnisse zu verifizieren. Denn gerade bei der Entwicklung, Analyse und Tests neuer Konzepte oder neu entwickelter

Tabelle 1.1.: Eine Auswahl der wichtigsten OFDM-Parameter für etablierte Kommunikationssysteme. Anhand dieser ist es leicht zu verdeutlichen, wie stark sich reelle Systeme in ihren Eigenschaften unterscheiden können.

Übertragungsstandard	DVB-T	IEEE 802.11a	LTE
Entwicklungsjahr	1997	1999	2006
Frequenzbereich (MHz)	470 - 862, 174 - 230	4915 - 5825	2600
Bandbreite (MHz)	8, 7, 6	20	1, 4, 3, 5, 10, 15, 20
Anzahl der Träger	2K Modus: 1705 8K Modus: 6817	48 (+4 Piloten)	1200
Trägermodulation	QPSK, 16-QAM, 64-QAM	BPSK, QPSK, 16-QAM, 64-QAM	QPSK, 16-QAM, 64-QAM
Typ. Symbollänge (μs)	2K Modus: 224 8K Modus: 896	3,2	66,67
Schutzintervall	1/4, 1/8, 1/16, 1/32	1/4	7%-25%
Nutzdatenrate (Mb/s)	0,576 - 1,152	6 - 54	3-300

Bauteile ist es hilfreich oder sogar notwendig, eine Vielzahl der möglichen Parameterkombinationen eines solchen Systems zu testen, um den Einfluss dieser auf die Komponenten und das Gesamtnetzwerk zu untersuchen. Hierbei soll das Ergebnis dieser Arbeit alles Nötige bieten, um mit einfachen Mitteln eine Entwurfsraumexploration der Parameter eines OFDM-Systems durchzuführen.

Die Fehlersuche ist ein weiterer Anwendungsfall, bei welchem das einfache Ändern der Parametereinstellungen hilfreich ist. Anhand der Resultate für unterschiedliche *Trägerabstände*, verschiedener *Modulationsarten* oder variierender Größe der *zyklischen Erweiterung* (Cyclic Prefix, CP) lassen sich eventuell Rückschlüsse auf das Problem einer Übertragung schließen. Auch die Korrekturfaktoren der *Kanalkorrektur* (Channel Equalization) eines OFDM-Empfängers können wertvolle Informationen über einen Übertragungskanal bereitstellen.

Des Weiteren kann der Einsatz verschiedener *Präambel- und Pilottonschemata*, oder auch das schrittweise Annähern an einen optimale Wert zur *Wertebegrenzung* (Clipping), bei der Datenausgabe durch ein solches Testsystems zu Erkenntnissen führen, welche anschließend in die Entwicklung von Systemen einfließen können, die speziell für einen bestimmten OFDM-Standard vorgesehen sind.

Der Kompromiss eines Echtzeit OFDM-Signalverarbeitungssystems, welches eine Kombination aus hohen Datenraten und hoher Flexibilität bietet, ist derzeit nicht verfügbar. Ein Grund hierfür dürfte darin zu finden sein, dass sich diese beiden Ziele in einer extremen Ausprägung gegenseitig ausschließen, da nur der Verzicht auf Flexibilität alle möglichen Optimierungen erlaubt. Allerdings ist auch kein System bekannt welches das Ziel verfolgt, den besten Kompromiss zu finden um genau diese Lücke zu füllen.

Die Verwendung bereits existierender IP-Sammlungen zur Signalverarbeitung wie beispielsweise das Xilinx Platform Studio ist nicht möglich, da diese nicht auf extreme Anwendungsfälle wie den hier angestrebten Datenraten im Gigabit-Bereich, ausgelegt sind. Auch sind nicht alle für OFDM notwendigen Algorithmen, beispielsweise zur Zeit- und Frequenzsynchronisation, in solchen Bibliotheken vorhanden.

1.2. Fragestellung und Zielsetzung

Für die erfolgreiche Verwirklichung des Ziels dieser Arbeit, ein Framework zu erstellen welches den einfachen Entwurf programmierbarer digitaler Signalverarbeitungssysteme für Hochgeschwindigkeits-OFDM erlaubt, beschäftigt sich diese Arbeit mit den folgenden fundamentalen Fragen:

- Was sind die Signalverarbeitungsschritte bzw. die Basisalgorithmen eines OFDM Systems? Wie sieht ein mögliches Hardware/Software Co-Design dieser Algorithmen aus?
- Welche Architektur eignet sich für solch ein OFDM-spezifisches Signalverarbeitungssystem, um einen guten Kompromiss zwischen Flexibilität und maximaler Datenrate zu erhalten?
- Was wird neben der Signalverarbeitung noch benötigt, um ein Testsystem zu erstellen?
- Welche Signalqualität wird benötigt, welche erreicht ein solches System?
- Welche Datenraten sind mit solch einem System maximal möglich?
- Wie kann man einen einfachen Entwurf und eine einfache Programmierung des Systems erreichen?

In den folgenden Kapitel sollen diese Fragen genauer betrachtet und beantwortet werden.

1.3. Aufbau der Arbeit

Zuerst werden in Kapitel 2 Grundlagen erläutert, welche benötigt werden, um die verschiedenen Entscheidungen und Entwicklungen dieser Arbeit nachzuvollziehen. Dabei handelt es sich zum einen um Grundlagen der digitalen Signalverarbeitung und der Kommunikationstechnik mit einem Fokus auf OFDM, zum andern werden Architekturen der digitalen Signalverarbeitung, insbesondere die hier verwendeten Field Programmable Gate Arrays, besprochen. Anschließend werden im Kapitel 3 verschiedene Architekturen bewertet, welche in jüngsten Veröffentlichungen verwendet wurden. Dabei wird sowohl ein Fokus auf eine hohe Datenrate als auch auf Flexibilität gelegt.

Den Kern der vorliegenden Arbeit stellen Kapitel 4 bis Kapitel 7 dar. Obwohl die Arbeit ein breites Anwendungsspektrum verfolgt, wird in Kapitel 4 zunächst ein Anwendungsszenario festgelegt, an welchem sich die Entwicklung orientiert. Anschließend erklärt dieses Kapitel zuerst die notwendigen Basisalgorithmen, welche ein OFDM-System realisieren muss. Sodann werden die prinzipiellen Designentscheidungen besprochen und der prinzipielle Aufbau eines auf dieser Architektur basierenden Systems dargestellt, bevor die Realisierung der einzelnen Signalverarbeitungsschritte mit Hilfe von Hardware Beschleunigern im Detail erläutert wird.

Im Folgekapitel 5 werden die mit Hilfe der entwickelten Hardware aufgebauten Prototypen eines OFDM-Übertragungssystems beschrieben. Danach werden verschiedene Experimente des OFDM-Senders und des OFDM-Empfängers mit und ohne optischer Übertragung erläutert und die Ergebnisse dieser Experimente dargestellt und bewertet.

In Kapitel 6 werden die Anforderungen für einen Digital Down Converter (DDC) dargelegt, wie er im Rahmen des Anwendungsszenarios zur OFDM-Vorverarbeitung benötigt wird. Anschließend wird die optimierte Implementierung sowie der Aufbau eines Prototyps besprochen. Das Kapitel schließt mit den durchgeführten Experimenten und deren Ergebnissen.

Kapitel 7 zeigt die Eingliederung der erstellten Hardware in ein Framework, um das System unversierten Anwendern leicht zugänglich zu machen. Dabei werden sowohl der Aufbau und der Entwurfsfluss eines Systems innerhalb des Frameworks dargestellt, als auch die Unterstützung für die Programmierung des entworfenen Systems.

Das letzte Kapitel 8 bewertet die entwickelte Architektur und die Ergebnisse der Experimente und gibt einen Ausblick auf mögliche Folgearbeiten.

2. Grundlagen

Um die Erläuterungen der nachfolgenden Kapitel sowie die Funktionsweise des entwickelten Systems nachvollziehen zu können, werden hier die dazu nötigen Grundlagen erörtert sowie auf vertiefende Literatur hingewiesen. Beginnend mit den allgemeinen Prinzipien der digitalen Datenübertragung und der Modulation in der Kommunikationstechnik, wird über einfache Modulationsarten die fortgeschrittene Modulationstechnik OFDM eingeführt, wobei sowohl auf deren prinzipielle Funktionsweise als auch auf ihre besonderen Eigenschaften eingegangen wird. Soweit nicht anders gekennzeichnet, stammen die Informationen zu diesem Abschnitt aus dem umfassenden Lehrwerk *Nachrichtenübertragung* von Karl-Dirk Kammeyer [46].

Anschließend werden die Merkmale und Besonderheiten von SRAM-basierten Field Programmable Gate Arrays (FPGAs) beschrieben, da die in dieser Arbeit entwickelte Architektur zur Realisierung auf rekonfigurierbarer Hardware dieses Typs hin optimiert ist. Insbesondere wird auf Details des Herstellers Xilinx eingegangen, da Bausteine dieses Herstellers in den aufgebauten Prototypen verwendet wurden. Für weiterführende Informationen sei auf [17] verwiesen.

Im letzten Unterkapitel werden abschließend noch die gängigen Hardware Architekturen und ihre Eigenschaften in Bezug auf Flexibilität und Leistungsfähigkeit beurteilt.

2.1. Digitale Datenübertragung und Modulation

Bei der digitalen Datenübertragung stellt eine digitale Datenquelle eine Folge von Symbolen *d(i)* zur Verfügung. Diese werden als diskrete Zahlenwerte aufgefasst wobei i den Zählindex der Daten angibt. Der Wertevorrat, auch Datenalphabet genannt, kann dabei zweiwertig (beispielsweise -1 und +1) oder höherwertig (beispielsweise -3, -1, +1, +3) sein. Bei einer M-wertigen Übertragung stellt jedes Symbol $\log_2(M)$ Bit dar.

Um diese Daten nun über einen analogen Kanal zu übertragen wird ein physikalischer Träger, beispielsweise schmale Spannungsimpulse, benötigt. Werden diese Impulssignale, auch Tiefpasssignale genannt, ohne eine zusätzliche Frequenzverschiebung direkt übertragen, spricht man von Basisbandübertragung.

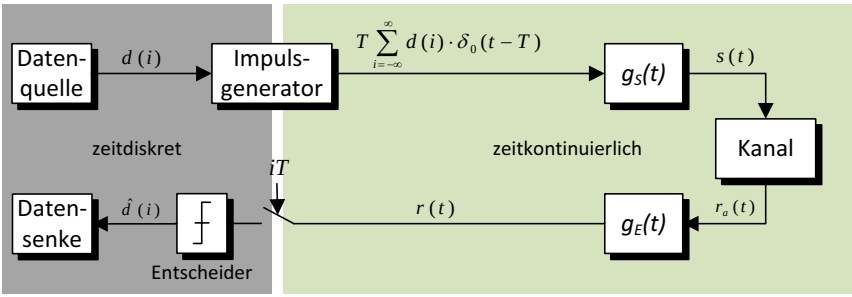


Abbildung 2.1.: Modell eines Datenübertragungssystems [46].

Abbildung 2.1 zeigt das Modell eines solches Übertragungssystems mit einem Tiefpasskanal, wobei T die Symboldauer, also den zeitlichen Abstand der Zahlenwerte der Symbolquelle, bezeichnet. Zur Bandbegrenzung und aus realisierungstechnischen Gründen werden im System ebenfalls ein Sendefilter mit der Impulsantwort $g_S(t)$ sowie ein Empfangsfilter mit einer Impulsantwort $g_E(t)$ benötigt. Durch diesen Tiefpassfilter $g_E(t)$ im Empfänger wird das Signal in seiner Bandbreite begrenzt. Dies ist wichtig, um bei der darauffolgenden Digitalisierung das *Nyquist-Shannon-Abtasttheorem* einzuhalten. Dieses besagt, dass ein auf die Frequenz f_{max} bandbegrenztes Signal mit einer Frequenz abgetastet werden muss, welche mindestens doppelt so hoch ist wie die Frequenz f_{max}, damit es nach der Digitalisierung wieder vollständig reproduzierbar ist. Für Details sei auf [46] und [94] verwiesen.

Der Begriff Modulation bezeichnet das Verändern (modulieren) eines Trägersignals anhand der Information des Quellensignals, womit diesem die Informationen des zu übermittelnden Signals aufgeprägt werden. So kann beispielsweise bei einer Bandpassübertragung ein Basisbandsignal als Quellensignal verwendet werden, um mit Hilfe eines Modulators eine höhere Trägerfrequenz zu modulieren, vergleiche Abbildung 2.2. Die Rückgewinnung des Quellensignals aus dem empfangenen Signal geschieht durch einen Demodulator und wird als Demodulation bezeichnet.

Ein Ziel der Trägermodulation kann es sein, das Sendesignal den Eigenschaften der Übertragungsstrecke anzupassen, indem beispielsweise eine Frequenz für das Trägersignal verwendet wird, welche sich im gegebenen Kanal besonders gut übertragen lässt. Zusätzlich ermöglicht die Modulation es, einen Kanal mehrfach zu belegen, indem mehrere Signale gleichzeitig übertragen werden. Dies ist möglich, indem verschiedene Übertragungssysteme unterschiedliche Trägerfrequen-

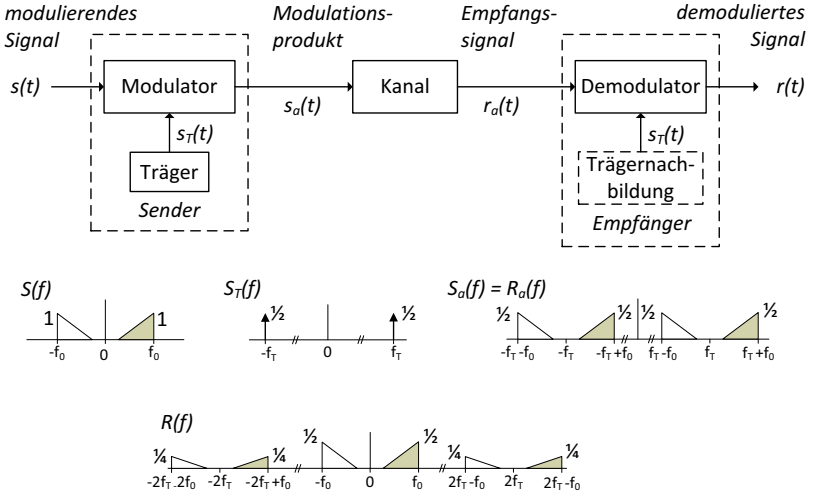


Abbildung 2.2.: Blockschaltbild einer Nachrichtenübertragung mit analoger Trägermodulation und den zugehörigen Spektren [94].

zen verwenden. Diese Träger sollten allerdings so weit auseinander liegen, dass es zu keiner gegenseitigen Beeinflussung kommt.

Modulation kann außerdem dazu verwendet werden, um die Datenrate zu erhöhen bzw. um Bandbreite einzusparen. So kann oftmals in modernen Kommunikationssystemen bei guten Übertragungsbedingungen die Datenrate dynamisch erhöht werden, indem eine höherwertige Modulation verwendet wird, welche allerdings wiederum ein höheres Signal-Rausch-Verhältnis benötigt.

2.1.1. Einträger Modulationsverfahren

Bei einfachen Modulationsarten wird normalerweise ein einzelner Träger moduliert. Dabei spricht man bei einem digitalen Quellensignal von einer digitalen Modulation, bei einem analogen Quellensignal von einer analogen Modulation, welche hier jedoch nicht weiter betrachtet werden soll.

Einfache Modulationsarten modulieren meist nur einen Parameter des Trägersignals. So wird bei der *Amplitudenumtastung* (Amplitude-shift keying, ASK) die Amplitude des Trägersignals verändert, bei der *Frequenzumtastung* (Frequency-shift keying, FSK) entsprechend die Frequenz des Trägersignals abgewandelt; bei

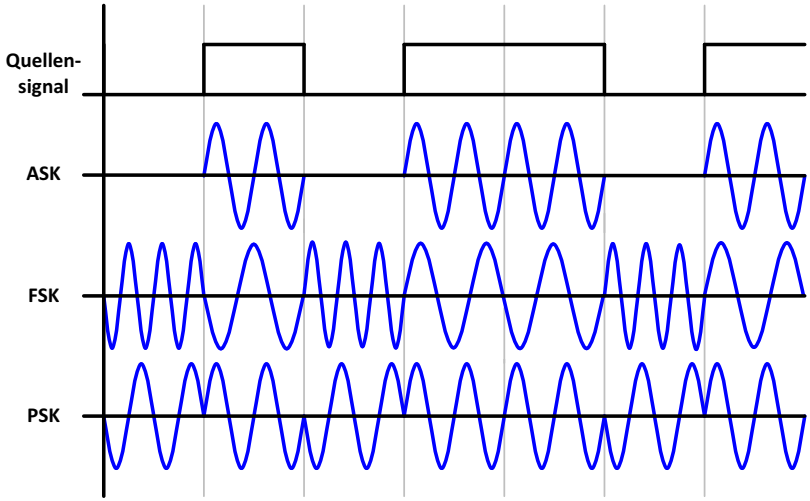


Abbildung 2.3.: Dargestellt ist ein digitales unipolares Quellensignal, eine Amplitudenumtastung (ASK), eine Frequenzumtastung (FSK) sowie eine Phasenumtastung (PSK) des Quellensignals [86].

der *Phasenumtastung* (Phase-shift keying, PSK) wird schließlich die Phase moduliert. In Abbildung 2.3 ist für jede der drei Modulationsarten ein Beispiel dargestellt, in welchem jeweils zwischen zwei Werten umgetastet wird, also pro Symbol ein Bit übertragen werden kann.

Technisch kann ein Modulator zur Amplitudenumtastung durch einen einfachen Mischer (z.B. einem Ringmodulator) realisiert werden. Die Demodulation einer Amplitudenumtastung kann im einfachsten Fall (Zweiseitenbandmodulation) mit einem einfachen Hüllkurvendemodulator bewerkstelligt werden. Dabei wird das Empfangssignal gleichgerichtet, durch einen Tiefpass gefiltert und durch einen Koppelkondensator von einer eventuell überlagerten Gleichspannung befreit.

Für komplexere Amplitudenumtastung, beispielsweise bei einer Einseitenbandmodulation, wird ein geeigneter Hilfsträger der möglichst gleichen Frequenz im Empfänger benötigt. Mit seiner Hilfe können entweder auch solche Signale gleichgerichtet werden, oder man kann das Informationssignal zurück in das Basisband mischen. Anschließend müssen hochfrequente Überlagerungen (z.B. durch die erste Harmonische des Trägers) entfernt werden. Für detaillierte Informationen zu den Modulatoren und den Demodulatoren der Amplitudenumtastung sowie

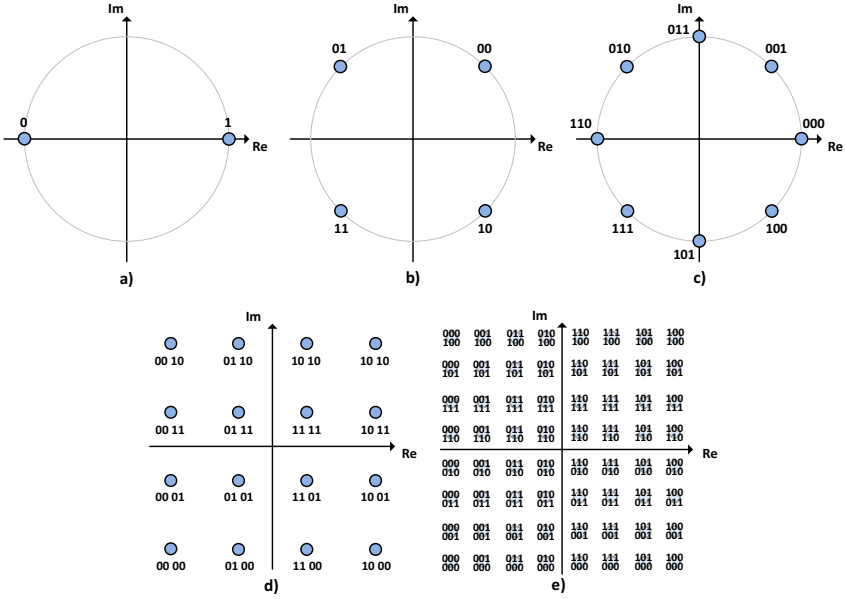


Abbildung 2.4.: Konstellationsdiagramme für a) BPSK, b) QPSK und c) 8-PSK sowie für die Quadraturamplitudenmodulationen d) 16-QAM und e) 64-QAM.

für entsprechende Informationen für die Frequenzumtastung und Phasenumtastung sei auf [86] verwiesen.

Soll die Datenrate erhöht oder soll Bandbreite eingespart werden, ist es möglich, pro Symbol mehr als ein Bit zu übertragen. Hierzu müssen jedoch mehr als zwei zulässige Werte pro Symbol existieren. Die 2-wertige Phasenumtastung nennt man binäre Phasenumtastung (Binary phase-shift keying, BPSK), eine 4-wertige Phasenumtastung nennt sich Quadraturphasenumtastung (Quadrature phase-shift keying, QPSK) und entsprechend existieren auch höherwertige Versionen, wie beispielsweise 8-PSK. In Abbildung 2.4 a) bis c) sind die Konstellationsdiagramme der genannten Phasenumtastungen, sowie Beispielzuordnungen für binäre Daten, dargestellt. Die Phase der Konstellationspunkte entspricht hier der Phasenlage des modulierten Trägersignals für diesen Wert.

Ebenso existieren natürlich auch höherwertige Modulationsformate für Amplitudenumtastung sowie für Frequenzumtastung. Die Nachteile höherwertiger Modulationsarten sind durch eine höhere Störanfälligkeit gegeben. So reichen bei-

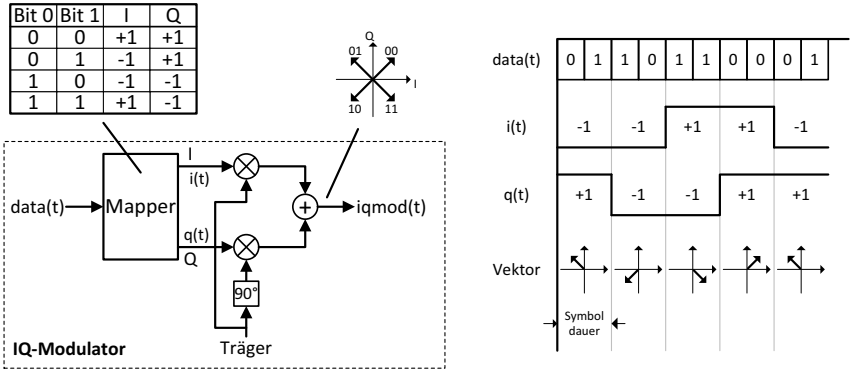


Abbildung 2.5.: Funktionsweise eines IQ-Modulators am Beispiel einer 4-wertigen Quadraturamplitudenmodulation [28].

spielsweise bei einer 8-PSK bereits Phasenungenauigkeiten von 22,5 Grad um einen empfangenen Wert einem falschen Sendewert zuzuordnen. Im Gegensatz hierzu können bei binärer Phasenumtastung theoretisch noch alle Phasenungenauigkeiten unter 90 Grad toleriert werden.

Für höherwertige Modulation besonders geeignet ist die Quadraturamplitudenmodulation (Quadrature Amplitude Modulation, QAM), da sie eine günstigere Verteilung mit möglichst viel Abstand der Werte untereinander im Konstellationsdiagramm ermöglicht. Sie kann als eine Kombination aus Amplitudenumtastung sowie Phasenumtastung verstanden werden, da verschiedene Werte unterschiedliche Amplituden oder unterschiedliche Phasen besitzen können.

Das Ergebnis einer 4-wertigen Quadraturamplitudenmodulation (4-QAM) ist in Abbildung 2.4 b) dargestellt und entspricht ebenso dem Ergebnis einer Quadraturphasenumtastung (Quadrature Phase-Shift Keying, QPSK). Abbildung 2.4 d) zeigt eine 16-wertige Quadraturamplitudenmodulation (16-QAM), eine 64-wertige Quadraturamplitudenmodulation (64-QAM) ist in Abbildung 2.4 e) zu sehen. Mit diesen Modulationsarten lassen sich vier beziehungsweise sechs Bits pro Symbol übertragen.

Zur Erzeugung quadraturamplitudenmodulierter Signale werden IQ-Modulatoren verwendet. Zuerst wird durch die Signalraum-Zuordnung (Mapping) im Mapper eines IQ-Modulators der eingehende Datenstrom korrekt auf zwei getrennte Signalpfade aufgeteilt, den I-Pfad und den Q-Pfad. Dabei steht I für *In-phase* und Q für *Quadrature Component*. Jeder Pfad realisiert eine eigenständige Amplitudenumtastung, wobei die In-phase-Komponente den unveränderten Träger modu-

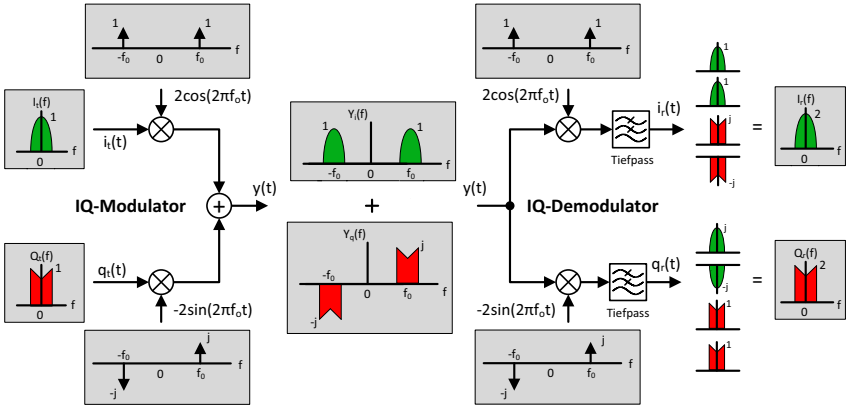


Abbildung 2.6.: Konzept der IQ-Modulation sowie der IQ-Demodulation mit Darstellung der entsprechenden Frequenzspektren.

liert. Die Quadratur-Komponente moduliert denselben Träger, allerdings mit einer um 90 Grad versetzten Phasenlage, wodurch I und Q orthogonal zueinander stehen. Für den Signalvektor des modulierten Symbols realisiert die I-Komponente den reellen Anteil, die Q-Komponente den imaginären Anteil. Das Sendesignal entsteht aus dem Summensignal des I-Pfades und des Q-Pfades. Abbildung 2.5 zeigt einen IQ-Modulator sowie die entsprechenden Signale für eine 4-wertige Quadraturamplitudenmodulation.

Zur Demodulation des empfangenen Signals wird ein IQ-Demodulator verwendet. Das empfangene Signal wird wieder in das Basisband herabgesetzt, indem es mit einem Hilfsträger der möglichst selben Frequenz des Sendeträgers gemischt wird. Dabei werden mittels zwei Mischern zwei Pfade erzeugt, der In-phase-Pfad unter Verwendung des unveränderten Hilfsträgers sowie der Quadraturphase-Pfad, dessen Hilfsträger wieder eine 90 Grad gedrehte Phasenlage besitzt. Durch diese Kombination eines spektral symmetrischen und eines spektral unsymmetrischen Hilfsträgers sowohl im Modulator als auch im Demodulator löschen sich im I-Pfad die Quadraturkomponenten, im Q-Pfad entsprechend die In-phase Signale aus. Hierdurch wird eine korrekte Rekonstruktion der In-phase und der Quadraturphase Signale möglich. Abbildung 2.6 veranschaulicht dieses Konzept anhand des Beispiels einer IQ-modulierten Übertragung. Zusätzlich werden die wichtigsten Spektren der Signale vor, während und nach der Übertragung aufgezeigt.

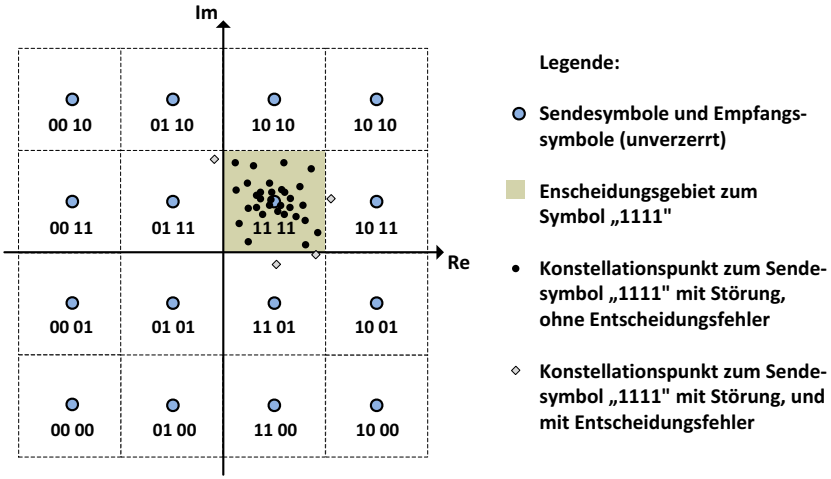


Abbildung 2.7.: Demapping: Zuordnung von empfangenen Konstellationspunkten anhand von Entscheidungsgebieten zu Sendesymbolen und zu den entsprechenden Bitfolgen am Beispiel von 16-QAM [94].

Zuletzt müssen die erhaltenen Punkte im Konstellationsraum noch durch einen Demapper zu einem gültigen Symbol und anschließend zu einer entsprechenden Bitfolge zugeordnet werden. Dabei wird jedes gültige Sendesymbol mit einem zugehörigen Entscheidungsgebiet versehen. Ein empfangener Konstellationspunkt wird nun dem Sendesymbol zugeordnet, in dessen Entscheidungsgebiet er sich befindet. Dabei können auf Grund von Störungen und einem daraus resultierenden zu großen Abstand zum Sendesymbol natürlich auch Entscheidungsfehler auftreten, vergleiche Abbildung 2.7.

2.1.2. Mehrträger Modulationsverfahren und OFDM

Erste Ideen zur Mehrträgerübertragung (Frequency Division Multiplexing, FDM) wurden bereits in den 50er Jahren formuliert, vgl. [66], es dauerte jedoch noch einige Jahre bis sich solche Techniken effizient nutzen ließen. Dabei kommen anstelle einer Trägerfrequenz mehrere verschiedene Trägerfrequenzen (auch Subträger genannt) zum Einsatz. Hierdurch ist es möglich, einen frequenzselektiven Kanal so in schmalbandige Subkanäle aufzuteilen, dass jeder der einzelnen Subkanäle annähernd nichtselektiv ist und eine Korrektur des Kanaleinflusses im Empfänger daher deutlich vereinfacht wird.

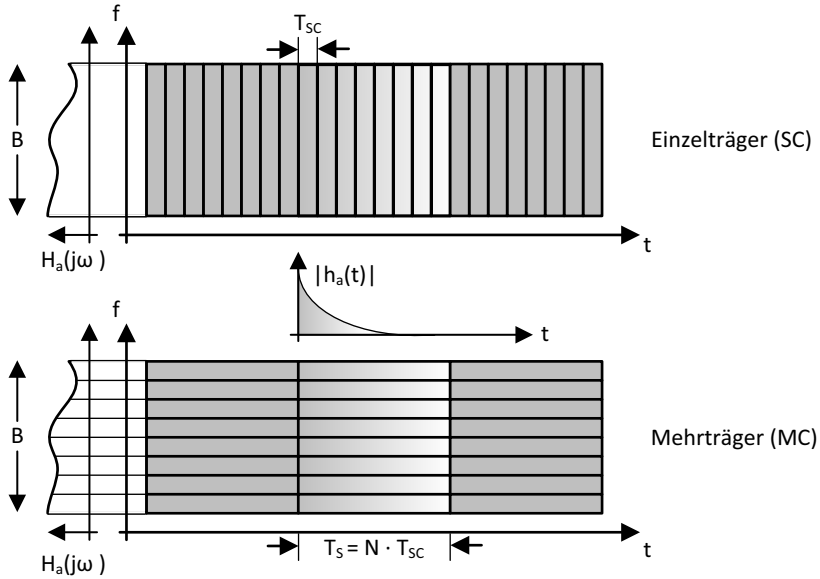


Abbildung 2.8.: Vergleich eines Einträgersignals (Single carrier, SC) mit einem N-Subträger Mehrträgersignal (Multi carrier, MC). Die Symboldauer T_S des Mehrträgersystems entspricht dabei der N-fachen Größe der Symboldauer T_{SC} des Einträgersystems. $h_a(t)$ stellt die Kanalimpulsantwort für diese Beispiel dar, $H_a(j\omega)$ den Kanalfrequenzgang [46].

Abbildung 2.8 verdeutlicht das Schema der *Einträgerübertragung* (Single carrier, SC) im Vergleich zu einer *Mehrträgerübertragung* (Multi carrier, MC) mit $N = 8$ Subträgern. Zum einen ist zu erkennen, dass die Bandbreite B der beiden Systeme prinzipiell gleich ist, die Breite des Trägers im Einträgersystems allerdings natürlich entsprechend größer ist als die Bandbreite eines einzelnen Trägers des Mehrträgersystems, im Beispiel also genau achtmal so groß.

Zum anderen ist die Symboldauer T_S des MC-Signals auch entsprechend acht mal größer als die Symboldauer T_{SC} des SC-Signals. Hierdurch verhält sich der Einfluss des Kanals anders: Während sich die Kanalimpulsantwort $h_{a(t)}$ beim Einträgersignal über mehrere Symbole erstreckt, überdeckt sie beim Mehrträgersignal nur einen Teil eines Symbols, wodurch die *Intersymbolinterferenz* (Intersymbol interference, ISI) deutlich reduziert ist.

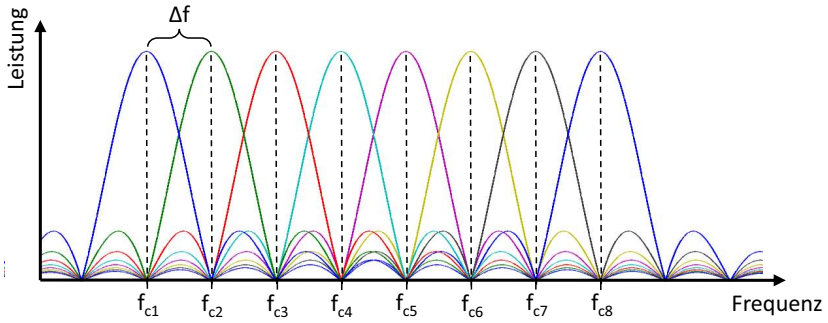


Abbildung 2.9.: OFDM Leistungsdichtespektrum. Beim Orthogonalen Frequenzmultiplexverfahren (OFDM) werden keine Schutzbänder zwischen den Trägern benötigt. Stattdessen kann ein Überlappen der Spektren toleriert werden, da die Träger orthogonal sind und somit im Maximum eines jeden Trägers die Spektren aller anderen Träger einen Nulldurchgang besitzen.

Analog verhält es sich mit dem Kanalfrequenzgang, welcher auf der linken Seite beider Signale jeweils angedeutet ist. Im Einträgerfall wirkt er über die gesamte Bandbreite B auf jedes Symbol, im Mehrträgerfall bilden sich pro Subkanal jeweils nahezu konstante Verläufe, wodurch quasi nichtselektive Verhältnisse vorliegen. Dadurch eignen sich Mehrträgerverfahren besonders dann, wenn ISI ein großes Problem darstellt, also zum Beispiel bei kabellosen Systemen mit Mehrwegausbreitung. Zusätzlich bieten FDM-Systeme einen weiteren Freiheitsgrad für Multiplexverfahren, siehe [46].

Ein Nachteil von FDM gegenüber Einträgerverfahren liegt in der benötigten Bandbreite. Da ein Überlappen der Subträgerspektren verhindert werden muss, werden zwischen den Frequenzkanälen *Schutzbänder* benötigt. Diese werden nicht direkt zur Datenübertragung verwendet und kosten daher zusätzlich Bandbreite. Die Schutzbänder sind allerdings nicht nur eine Hilfe zur Vermeidung von *Intercarrier Interferenzen* (Intercarrier interference, ICI) sondern entspannen auch die Anforderungen an die Flankensteilheit der Filter, welche standardmäßig zum Auftrennen der Kanäle beim Frequenzmultiplexverfahren verwendet werden.

Abhilfe für den Einsatz von Schutzbändern schafft das orthogonale Frequenzmultiplexverfahren (Orthogonal Frequency-Division Multiplexing, OFDM). Wie in Abbildung 2.9 zu sehen sind hier keine Schutzbänder notwendig. Die Träger werden so dicht aneinander gereiht, dass sich deren Spektren sogar überlappen. Eine fehlerfreie Auswertung ist dennoch möglich. Dies liegt daran, dass die Sub-

träger orthogonal zueinander sind, was bedeutet, dass im Maximum eines jeden Trägers alle anderen Subträger einen Nulldurchgang besitzen und somit an dieser Stelle keinen Einfluss auf den Träger nehmen. Dies ermöglicht es OFDM prinzipiell die Bandbreiten-Effizienz eines Einträgerverfahrens zu erreichen.

Ein weiterer Nachteil der Mehrträgerverfahren ist der erhöhte Aufwand. Um zu vermeiden, dass man für jeden Träger einen eigenen Sender und Empfänger bereitstellen muss, bedient man sich der digitalen Signalverarbeitung. In Abbildung 2.10 links ist ein Modell zur Erstellung des Basisbandsignals eines einfachen Frequenzmultiplexverfahrens mit N Trägern gezeigt. Durch eine Seriell zu Parallel Wandlung (S/P) werden jeweils $\log_2(M)$ Bit der Quelle in N parallele Ströme geformt. Nach dem Mapping in ein M-wertiges Alphabet ($d_0(i)$-$d_{N-1}(i)$) erfolgt die Bandbegrenzung durch N identische Tiefpässe mit den reellen Impulsantworten $g_S(t)$.

Durch diese Seriell-zu-Parallel Wandlung ergibt sich die N-fache Symboldauer. Sei T_{bit} die Dauer eines Quellbits so beträgt die Dauer des resultierenden Mehrträgersymbols ($s_a(t)$)

$$T_S = N \cdot ld(M) \cdot T_{bit}. \tag{2.1}$$

Die Nyquist-Bandbreite der benötigten Sende-Tiefpässe ergibt sich somit zu $1/T_S$.

Die parallelen Signale werden nach den Tiefpässen den Subträgern f_0-f_{N-1} aufmoduliert. Aufsummiert ergibt sich daraus das Basisbandsignal des FDM-Senders, welches anschließend durch einen geeigneten Mischer, für eine Bandpass-Übertragung auf eine entsprechende Trägerfrequenz gesetzt werden kann.

Sollen nun äquidistante Subträger mit den Abständen $1/T_S$ eingesetzt werden, folgt für die Subträgerfrequenzen

$$f_n = \frac{n}{T_S}, \quad \text{für } n = 0, \cdots, N-1. \tag{2.2}$$

Hieraus ergibt sich für das Mehrträgersignal in Basislage

$$s_{MC}(t) = T_S \sum_{n=0}^{N-1} \sum_{i=-\infty}^{\infty} d_n(i) g_S(t - iT_s) e^{j2\pi f_n t}, \tag{2.3}$$

wobei der Skalierungsfaktor T_S für ein dimensionsloses Signal sorgt.

Geht man von Sendefiltern mit unendlich steiler Flanke aus, kann für ihre Impulsantworten ein Rechteck der Dauer T_S verwendet werden

$$g_S(t) = \begin{cases} 1/T_S & \text{für } 0 \leq t < T_S \\ 0 & \text{sonst,} \end{cases} \tag{2.4}$$

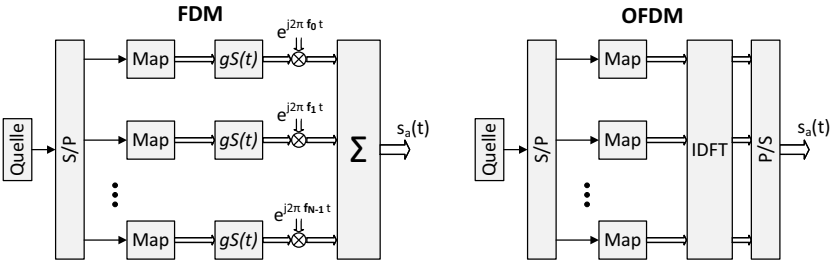


Abbildung 2.10.: Falls bei Frequenzmultiplexverfahren orthogonale Träger verwendet werden (OFDM) kann man die parallele Anordnung von Modulatoren im Sender durch eine inverse diskrete Fourier Transformation ersetzen.

woraus für das Mehrträgersignal folgt

$$s_{MC}(t) = \sum_{n=0}^{N-1} d_n(i) \cdot e^{j2\pi f_n t}, \quad \text{für} \quad iT_S \leq t < (i+1)T_S. \tag{2.5}$$

Für eine zeitdiskrete Darstellung mit der Abtastfrequenz

$$F_A = \frac{N}{T_S} \tag{2.6}$$

gilt für das Mehrträgersignal

$$s_{MC}(k' \frac{T_S}{N}) = \sum_{n=0}^{N-1} d_n(i) \cdot e^{j2\pi nk'/N} \quad \text{für} \quad iN \leq k' < (i+1)N - 1. \tag{2.7}$$

Dies entspricht, abgesehen vom Skalierungsfaktor $1/N$ der inversen diskreten Fourier-Transformation (IDFT). Für weitere Details sowie für eine Überprüfung der Orthogonalität solch eines Mehrträgersignals (2.7) sei auf [46] verwiesen. Wie in Abbildung 2.10 zu sehen ist, kann also bei OFDM anstelle von parallelen Modulatoren die inverse diskrete Fourier-Transformation verwendet werden, welche sehr effizient mittels der *schnellen Fourier-Transformation* realisiert werden kann.

2.1.3. Schnelle Fourier-Transformation

Die schnelle Fourier-Transformation (Fast Fourier Transform, FFT) bildet ein zentrales Element eines OFDM Systems. Sie fand starke Verbreitung durch die Veröffentlichung von James Cooley und John W. Tukey [21] und ermöglicht durch die Ausnutzung von Redundanzen eine effiziente und schnelle Realisierung der diskreten Fourier-Transformation (DFT). Falls nicht entsprechend gekennzeichnet, können die Informationen diese Kapitels in [19] nachgeschlagen werden.

In Gleichung 2.8 wird die Definition der DFT aufgezeigt. Für die komplexen Drehfaktoren W_N^{nk} stößt man in der Literatur, auch im Deutschen, oft auf den englischen Begriff „twiddle factor“.

$$\begin{aligned} X_n &= \sum_{k=0}^{N-1} x_k e^{-j2\pi\frac{nk}{N}} \\ &= \sum_{k=0}^{N-1} x_k W_N^{nk}, \quad \text{für } n = 0, \cdots, N-1 \end{aligned} \tag{2.8}$$

Die Definition der inversen DFT ist in Gleichung 2.9 gegeben. Es ist leicht zu erkennen, dass sie sich von der DFT nur durch den Faktor 1/N und durch das Vorzeichen des Drehfaktors unterscheidet.

$$\begin{aligned} x_k &= \frac{1}{N} \sum_{n=0}^{N-1} X_n e^{j2\pi\frac{nk}{N}} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} X_n W_N^{-nk} \end{aligned} \tag{2.9}$$

Wendet man die Beziehungen 2.10 und 2.11 sowie die komplexe Konjunktion auf Gleichung 2.9 an, dann zeigt sich, dass sich die inverse DFT mittels der DFT berechnen lässt, vergleiche Gleichung 2.12.

$$-j \cdot j = -j^2 = -(-1) = 1 \tag{2.10}$$

$$-j^* = j \tag{2.11}$$

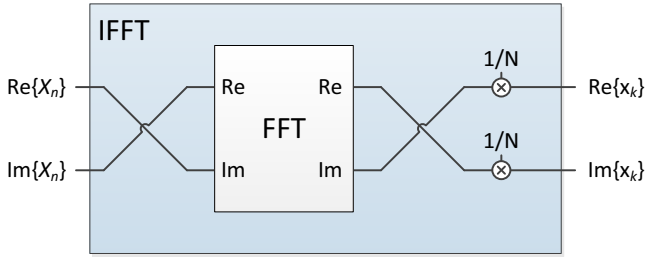


Abbildung 2.11.: Bildung einer inversen FFT aus der Berechnung einer FFT.

$$\begin{aligned} x_k &= \frac{1}{N} \sum_{n=0}^{N-1} X_n e^{j2\pi \frac{nk}{N}} \\ &= \frac{1}{N}\, j \sum_{n=0}^{N-1} -j\, X_n e^{j2\pi \frac{nk}{N}} \\ &= \frac{1}{N}\, j \left(\sum_{n=0}^{N-1} -j\, X_n^* e^{j2\pi \frac{nk}{N}} \right)^* \\ &= \frac{1}{N}\, j\, (DFT\,\{j\, X_n^*\})^* \end{aligned} \tag{2.12}$$

Um die inverse DFT mit Hilfe der DFT zu berechnen, muss die Eingangsfolge komplex konjugiert und mit der imaginären Zahl j multipliziert werden. Dies kann durch das Vertauschen von Real- und Imaginärteil erreicht werden, siehe Gleichung 2.13. Anschließend ist diese Operation ein zweites mal mit den Ergebnissen der DFT durchzuführen. Es folgt eine Skalierung mittels einer Division durch N. Die selbe Beziehung gilt auch für die FFT und die inverse FFT. Abbildung 2.11 verdeutlicht den Zusammenhang noch einmal.

$$jX^* = j\,(Re(X) + j\,Im(X))^* = j\,(Re(X) - j\,Im(X)) = Im(X) + j\,Re(X) \tag{2.13}$$

Für die klassische FFT nach Cooley und Tukey, auch Radix-2-Algorithmus genannt, muss die Anzahl an Stützstellen eine Potenz von zwei sein. Ist dies der Fall, lässt sich die DFT in zwei DFTs mit der halben Anzahl an Stützstellen aufteilen. Dies kann so lange fortgeführt werden, bis eine Stützstellenanzahl von 2 erreicht ist und daher nur noch zwei Werte miteinander verrechnet werden müs-

sen. Die Teilergebnisse der aufgeteilten DFTs werden nun solange wieder vereint, bis die gesamte Transformation abgeschlossen ist.

Es existieren weitere Algorithmen mit anderen Kernkomponenten der Zerlegung, beispielsweise der Radix-4-Algorithmus oder Split-Radix-Algorithmus. Sie bieten meist noch effizientere Wege, bringen teilweise aber auch weitere Einschränkungen. Außerdem gibt es zwei Arten der Aufteilung einer DFT. Während Variante eins die Zerlegung in der Zeitebene durchführt (Decimation in Time, DIT) findet die Zerlegung bei Variante zwei in der Frequenzebene statt (Decimation in Frequency, DIF). Im Folgenden wird der klassisch DIT Radix-2-Algorithmus näher beleuchtet. Weitere Informationen zu dieser und zu anderen Varianten finden sich beispielsweise in [19].

2.1.3.1. Decimation-In-Time Radix-2-FFT

Nach Cooley und Tukey wird die DFT-Ausgangsformel 2.8 zunächst nach geraden und ungeraden Indizes getrennt.

$$\begin{aligned} X_n &= \sum_{k=0}^{N-1} x_k e^{-j\frac{2\pi}{N}nk} \\ &= \sum_{k=0}^{\frac{N}{2}-1} x_{(2k)} e^{-j\frac{2\pi}{N}n(2k)} + \sum_{k=0}^{\frac{N}{2}-1} x_{(2k+1)} e^{-j\frac{2\pi}{N}n(2k+1)} \\ &= \sum_{k=0}^{\frac{N}{2}-1} x_{(2k)} e^{-j\frac{2\pi}{\frac{N}{2}}nk} + e^{-i\frac{2\pi}{N}n} \sum_{k=0}^{\frac{N}{2}-1} x_{(2k+1)} e^{-j\frac{2\pi}{\frac{N}{2}}nk} \quad \text{mit } e^{-i\frac{2\pi}{N}n} = W_N^n \\ &= \frac{N}{2} DFT\{x_{(2k)}\} + W_N^n \cdot \frac{N}{2} DFT\{x_{(2k+1)}\}, \quad \text{für } n = 0, \ldots, N-1 \end{aligned} \tag{2.14}$$

Aus Gleichung 2.14 wird also auch mathematisch ersichtlich, wie sich eine DFT mit N-Stützstellen in 2 DFTs mit N/2 Stützstellen teilen lässt, wobei die DFT der ungeraden Indizes noch mit dem Drehfaktor W_N^n multipliziert werden muss. Da sich nach demselben Prinzip auch die N/2 DFTs zerlegen lassen ist es möglich, in der Zeitebene zerlegte DFTs der Größe zwei so lange zu kombinieren, bis man eine DFT der Größe N erhält. Abbildung 2.12 veranschaulicht diesen Sachverhalt wobei die obere Hälfte der zerlegten Stufen dabei die geraden Indizes und die untere Hälfte die ungeraden Indizes realisiert.

Um die Funktionsweise besser zu verstehen, wird Gleichung 2.14 mit $y(k) = x(2k)$ und $z(k) = z(2k+1)$ in zwei Teilprobleme gegliedert:

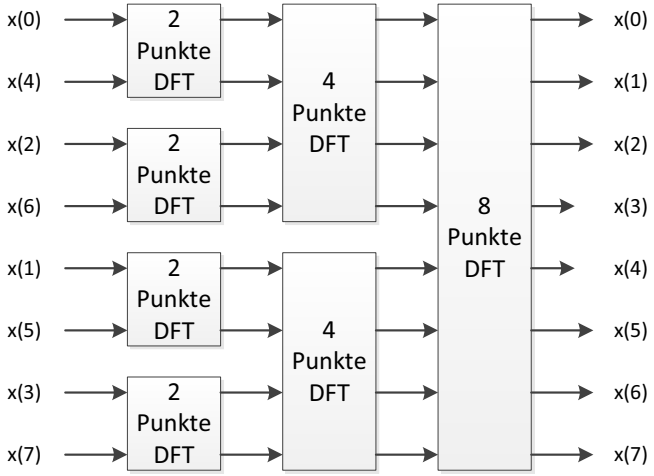


Abbildung 2.12.: Aufteilung einer 8-Punkte DFT in der Zeitebene (DIT).

$$Y_n = \sum_{k=0}^{\frac{N}{2}-1} y_k W_{\frac{N}{2}}^{nk}, \quad \text{für } n = 0, \ ..., \ \frac{N}{2} - 1 \tag{2.15}$$

$$Z_n = \sum_{k=0}^{\frac{N}{2}-1} z_k W_{\frac{N}{2}}^{nk}, \quad \text{für } n = 0, \ ..., \ \frac{N}{2} - 1 \tag{2.16}$$

Liegen die Ergebnisse dieser Teilprobleme vor, lässt sich das Gesamtproblem für die ersten N/2 Indizes lösen:

$$X_n = Y_n + W_N^n Z_n, \quad \text{für } n = 0, \ ..., \ \frac{N}{2} - 1 \tag{2.17}$$

Unter Einbeziehung von $W_N^{n+\frac{N}{2}} = -W_N^n$ und $W_N^N = 1$ folgt für die restlichen N/2 Terme:

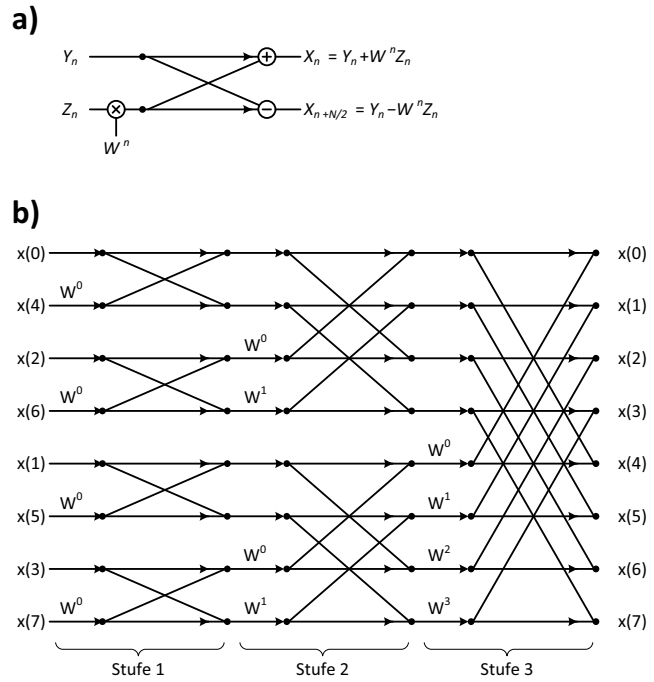


Abbildung 2.13.: a) Radix-2 Butterfly. b) Radix-2 DIT-FFT Implementierung mit 8 Stützstellen.

$$\begin{aligned} X_{(n+\frac{N}{2})} &= \sum_{k=0}^{\frac{N}{2}-1} y_k W_{\frac{N}{2}}^{(n+\frac{N}{2})k} + W_N^{n+\frac{N}{2}} \sum_{k=0}^{\frac{N}{2}-1} z_k W_{\frac{N}{2}}^{(n+\frac{N}{2})k} \\ &= Y_n - W_N^n Z_n, \quad \text{für } n = 0, \ldots, \frac{N}{2} - 1 \end{aligned} \tag{2.18}$$

Zusammen bilden Gleichung 2.17 und 2.18 den Cooley-Tukey Butterfly, auch Radix-2 Butterfly genannt, siehe Abbildung 2.13 a). In Abbildung 2.13 b) ist eine entsprechende Struktur für eine DIT-FFT mit 8 Stützstellen dargestellt.

Es ist leicht zu erkennen, dass sich so jede beliebige FFT mit $log_2(N)$ Stufen zu je N/2 Butterfly Operationen berechnen lässt. Tabelle 2.1 verdeutlicht die potentiel-

Tabelle 2.1.: Vergleich zwischen DFT und FFT anhand eines Beispiels mit 64 Stützstellen [26].

Operation	DFT	FFT
Komplexe Addition	$N^2 - N = 4032$	$N \cdot log_2(N) = 384$
Komplexe Multiplikation	$N^2 = 4096$	$\frac{N}{2} \cdot log_2(N) = 192$

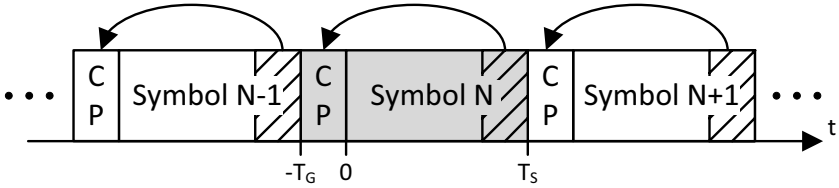


Abbildung 2.14.: Bei der zyklischen Erweiterung (Cyclic Prefix, CP) wird der hintere Symbolabschnitt des nachfolgenden Symbols als Schutzintervall verwendet.

len Einsparungen der FFT gegenüber der DFT an Additionen und Multiplikationen am Beispiel für 64 Stützstellen. Zu beachten ist, dass die Eingangsdaten nicht in normaler Zählreihenfolge verwendet werden, sondern *bitreversed*. Für weitere Details sei auf [19] verwiesen.

2.1.4. Schutzintervall und zyklische Erweiterung

In Kapitel 2.1.2 wurde bereits beschrieben, weshalb Mehrträgerverfahren prinzipiell weniger anfällig für Intersymbol-Interferenzen sind. Dennoch führt bei einem frequenzselektiven Kanal der Einschwingvorgang zu Beginn eines Symbols zum Verlust der Orthogonalität der Träger (Intercarrier-Interferenz, ICI), der Ausschwingvorgang überlagert (wenn auch nur teilweise) das nachfolgende Symbol und führt zu ISI. Diesen Problemen kann mittels eines *Schutzintervalls* der Länge T_G zwischen den Symbolen entgegengewirkt werden, wobei T_G größer gleich der Kanalimpulsantwort sein muss.

Im Normalfall wird bei OFDM anstatt eines einfachen Schutzintervalls eine *zyklische Erweiterung* (Cyclic Prefix, CP) verwendet. Gegenüber einer *Guardlücke*, bei welcher während des Schutzintervalls nichts gesendet wird, wird der hintere Symbolabschnitt des nachfolgenden Symbols übermittelt, vergleiche Abbildung 2.14. Zum einen kann dies Vorteile hinsichtlich der Zeitsynchronisation bieten, da je nach Größe der zyklischen Erweiterung Ungenauigkeiten in der Synchronisie-

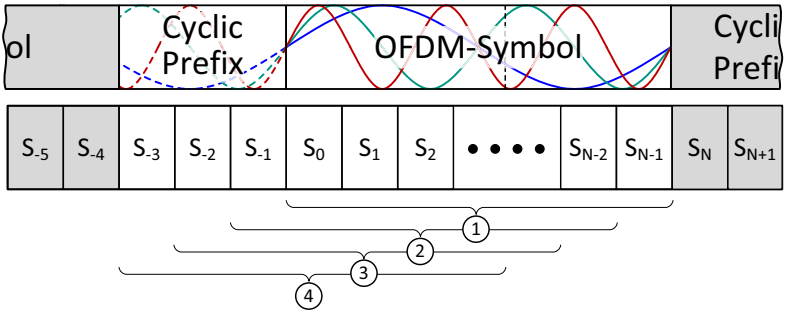


Abbildung 2.15.: Durch die zyklische Erweiterung hat man im Idealfall mehrere Möglichkeiten für eine zeitliche Synchronisierung, da ein Synchronisieren unter Nutzung der zyklischen Erweiterung nur zu einem konstanten Phasenfehler führt, welcher später im Empfänger ohne Probleme ausgeregelt werden kann. Die vorderen Samples der zyklischen Erweiterung sind allerdings Normalfall durch Intersymbol-Interferenzen gestört.

rung toleriert werden können (siehe Kapitel 2.1.5). Zum anderen beinhaltet die Verwendung des Guardlücke Verfahrens einige Nachteile, zum Beispiel hinsichtlich der automatischen Amplitudenregelung, vgl. [46].

Jegliche Art von Schutzintervall reduziert allerdings die Bandbreiten-Effizienz des Übertragungssystems. Vernachlässigt man spektrale Ausschwinger und nimmt für die Übertragungsbandbreite N/T_S so ergibt sich die Bandbreite-Effizienz zu

$$\beta = \frac{\text{Symbolrate}}{\text{Bandbreite}} = \frac{1}{1 + T_G/T_S}. \tag{2.19}$$

Daher lassen sich Schutzintervalle aufgrund der wesentlich höheren Symboldauer von Mehrträgerverfahren effizienter realisieren als bei Einträgerverfahren.

2.1.5. Synchronisation

Wie in allen digitalen Kommunikationssystemen spielt auch bei OFDM die Synchronisierung eine wichtige Rolle. Dabei findet eine solche sowohl im Zeitbereich als auch im Frequenzbereich statt. Die zeitliche Synchronisierung ist notwendig, um die Grenzen der OFDM Symbole bestimmen zu können, diese von der zyklischen Erweiterung zu befreien und dadurch die korrekten Samples für

die nachfolgende Fourier Transformation zu verwenden. Bei einer Übertragung mit N Subträgern hat man für die Auswahl des N-Sample großen FFT-Fensters, aufgrund der zyklischen Erweiterung, mehrere Möglichkeiten, vergleiche Abbildung 2.15. Allerdings ist zu beachten, dass je nach Kanalantwort und Größe der zyklischen Erweiterung, einige Samples der zyklischen Erweiterung durch Intersymbol-Interferenzen gestört sein können (vgl. Abschnitt 2.1.4).

Neben der Synchronisation im Zeitbereich spielt bei OFDM die Synchronisation im Frequenzbereich eine besonders wichtige Rolle, da bei Abweichungen der korrekten Trägerfrequenzen beim Empfänger immer auch ein Verlust der Orthogonalität der Subträger untereinander entsteht, was schließlich zu Inter-Carrier-Interferenzen führt, vgl. Abbildung 2.9.

Eine Synchronisation kann mehrstufig, beispielsweise zweistufig, aufgebaut sein. Hierbei findet zuerst eine grobe Synchronisation statt, um anschließend mit angepassten Parametern oder Verfahren eine Feinabstimmung vorzunehmen. Die verschiedenen Synchronisationstechniken nutzen meist entweder Pilotträger beziehungsweise eine Präambel zur Synchronisation (vergleiche [80] oder [65]) oder sie verwenden die Redundanz der zyklischen Erweiterung (siehe [10]). Ein Pilottträger ist ein Subträger welcher lediglich Pilottöne, also bekannte Referenzwerte sendet.

2.1.5.1. Synchronisation im Zeitbereich

Die Synchronisation im Zeitbereich findet über Metriken statt, welche häufig über Korrelationen gebildet werden. So kann beispielsweise die Kreuzkorrelation einer Präambel mit einer im Empfänger gespeicherten Version dieser Präambel verwendet werden. Alternativ kann zum Beispiel die Autokorrelation einer sich wiederholenden Präambel genutzt werden (z.B. IEEE 802.11a, vgl. [34]) oder es kommt direkt die Autokorrelation des Datensignals zum Einsatz, indem die Korrelation der zyklischen Erweiterung mit dem entsprechenden Bereich im eigentlichen Symbol ausgenutzt wird (z.B. DVB-T vgl. [34]).

Da die Signale der Kreuzkorrelation aufgrund des Kanaleinflusses stark voneinander abweichen können, hat sich in den meisten Systemen die Autokorrelation einer Präambel durchgesetzt. Ähnliches gilt für den Vergleich der zyklischen Erweiterung aufgrund der Intersymbol-Interferenzen. Daher soll im Folgenden lediglich dieses Konzept näher erläutert werden. Für tiefer gehende Informationen, speziell zu den anderen Verfahren, sei auf [34] und [56] verwiesen.

Obwohl es mehrere Studien und Anpassungen zur Optimierung dazu gibt, wie beispielsweise in [62], soll hier das grundlegende Konzept der Präambel basierten Synchronisation wie in [80] vorgestellt werden. Dabei wird eine sich im Zeitbereich einmal wiederholende N-Samples große Sequenz verwendet. Die Präambel

zur Synchronisation ist daher insgesamt $2N$ groß. Der Empfänger berechnet nun die Autokorrelation P des Eingangssignals mit einer Verschiebung τ der Größe der Sequenz zu

$$\begin{aligned} P(d) &= \sum_{m=0}^{N-1} (r_{d+m}^{*} r_{d+m+\tau}) \\ &= \sum_{m=0}^{N-1} (r_{d+m}^{*} r_{d+m+N}). \end{aligned} \tag{2.20}$$

Dabei stellt d den zeitlichen Index der digitalen Samples dar. Parallel zur Berechnung der Autokorrelation wird die Energie R des Signals gebildet

$$R(d) = \sum_{m=0}^{N-1} |r_{d+m+N}|^2. \tag{2.21}$$

Diese zwei Werte werden dann zu einer Metrik M verrechnet

$$M = \frac{|P(d)|^2}{(R(d))^2}. \tag{2.22}$$

Dabei wird der eigentliche Verlauf der Metrik durch die Autokorrelation im Zähler vorgegeben. Der Energie-Ausdruck im Nenner dient lediglich der Normierung. Er verhindert, dass absolute Unterschiede in der empfangenen Leistung den Verlauf der Gesamtmetrik zu stark beeinflussen. Der genaue Verlauf der Metrik hängt von der Struktur der Präambel ab. In Abbildung 2.16 ist ein qualitativer Beispielverlauf für eine [80] entsprechende Präambel zu sehen. Hier bildet sich ein Plateau von der Größe der zyklischen Erweiterung aus, mit dessen Hilfe dann die Symbolgrenzen bestimmt werden können.

2.1.5.2. Synchronisation im Frequenzbereich

Zur Synchronisation im Frequenzbereich wird in [80] dieselbe Präambel verwendet wie zur Synchronisation der Zeit. Da bekannt ist, dass sich alle Samples nach N-Schritten wiederholen, kann man ein Sample der ersten Sequenz nehmen und dieses mit dem entsprechenden Sample der zweiten Sequenz vergleichen. Geht man von einem rauschfreien Kanal aus, welcher hinreichend zeitinvariant ist, so sollte man für beide Samples denselben Wert bzw. dieselbe Phasenlage erhalten. Besteht zwischen zwei entsprechenden Samples der Samplerate f_{Sample} eine Pha-

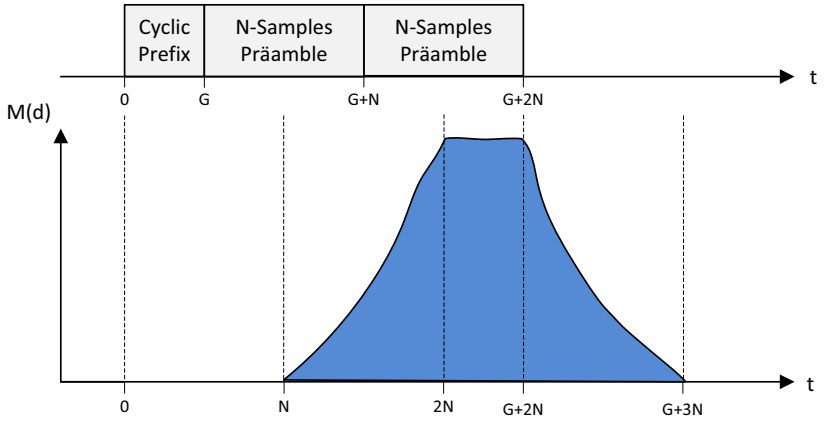


Abbildung 2.16.: Aufbau einer Präambel (oben) nach Schmidl-Cox [80] und qualitativer Verlauf der entsprechenden Metrik (unten). Das sich bildende Plateau hat dabei die Länge der zyklischen Erweiterung.

sendifferenz ϕ, so lässt sich die Frequenzabweichung durch Gleichung 2.23 errechnen.

Zu Beachten ist jedoch, dass hierbei eine korrekte Schätzung nur möglich wird, wenn der Betrag der Phasendifferenz kleiner als 180 Grad ist. Da diese Differenz stark vom Abstand N der betrachteten Samples abhängt, lässt sich über diesen Parameter die Auflösung der Schätzung steuern. Auch hier können mehrstufige Verfahren zur Korrektur der Frequenz eingesetzt werden. Zuerst wird durch eine erste Präambel mit kleinem N die Frequenz grob eingestellt, anschließend kann mittels einer zweiten Präambel mit größerem N eine Feinjustierung vorgenommen werden.

$$\Delta f = \frac{\phi \cdot f_{Sample}}{360° \cdot N}. \tag{2.23}$$

Um den durch Rauschen verursachten Fehler solch einer Frequenzschätzung zu minimieren, wird die Differenzfrequenz normalerweise nicht anhand eines einzigen Paares von Abtastwerten geschätzt, stattdessen wird der Mittelwert mehrerer Paare verwendet. Zur Bildung dieses Mittelwertes $\overline{\phi}$ kann man die zuvor berechnete Autokorrelation aus Gleichung 2.20 verwenden:

$$\overline{\phi} = \angle \{P(d)\}. \tag{2.24}$$

Wurde der Frequenzversatz errechnet, kann dieser beispielsweise mit der Hilfe eines numerisch gesteuerten Oszillators (Numerically controlled oscillator, NCO) und eines digitalen Quadratur=Mischers korrigiert werden.

2.1.6. Kanalschätzung und Kanalkorrektur

Abgesehen von einer inkohärenten Detektion, wie sie beispielsweise mittels differentieller Modulation der Trägerfrequenzen möglich wäre, muss für den korrekten Empfang der Daten die Phase der Träger wiederhergestellt werden. Hierzu werden Informationen über den Übertragungskanal benötigt, welche durch eine Kanalschätzung ermittelt werden können, siehe [9]. Dabei ist es möglich, den Kanal mit Hilfe einer Präambel oder mit Hilfe von Pilotträgern zu schätzen.

2.1.6.1. Präambelbasierte Kanalschätzung

Bei Systemen, die eine Übertragung blockweise vornehmen (z.B. IEEE 802.11a) und bei denen der Kanal ausreichend zeitinvariant ist, eignet sich der Einsatz einer Präambel zur Kanalschätzung. Die Präambel befindet sich am Anfang eines jeden OFDM-Blocks und enthält eine Trainigssequenz P welche dem Empfänger bekannt ist. Durch den Vergleich der empfangenen Trainingssequenz $\tilde{P}$ mit der gesendeten Trainingssequenz wird die Kanalschätzung wie in Gleichung 2.25 je einmal pro Block durchgeführt. Der Kehrwert dieser Schätzung wird dementsprechend für die darauffolgenden Datensymbole des OFDM-Blocks als Korrekturfaktor verwendet.

Abbildung 2.17 zeigt das Pilotenschema bei IEEE 802.11a, bei welchem die ersten zwei Symbole aus einer Präambel bestehen. Das erste dieser Symbole dient der Zeitsynchronisierung und der groben Synchronisation im Frequenzbereich. Mit Hilfe des zweiten Präambelsymbols wird dann feinjustiert sowie eine Kanalschätzung durchgeführt. Die nachfolgenden Pilotträger werden bei IEEE 802.11a lediglich zur Aktualisierung der Frequenzsynchronisation verwendet.

$$H = \frac{\tilde{P}}{P} \tag{2.25}$$

2.1.6.2. Piloten basierte Kanalschätzung

Für kontinuierliche Signalübertragung und für blockbasierte Systeme, deren Wiederholungszyklus sich nur schwer an die Zeitvarianz des Kanals anpassen lässt, eignet sich eine Nachführung der Entzerrung. Dies kann mit Hilfe von Pilotträgern geschehen. Pilotträger sind Subträger, welche nicht zur Datenübertragung

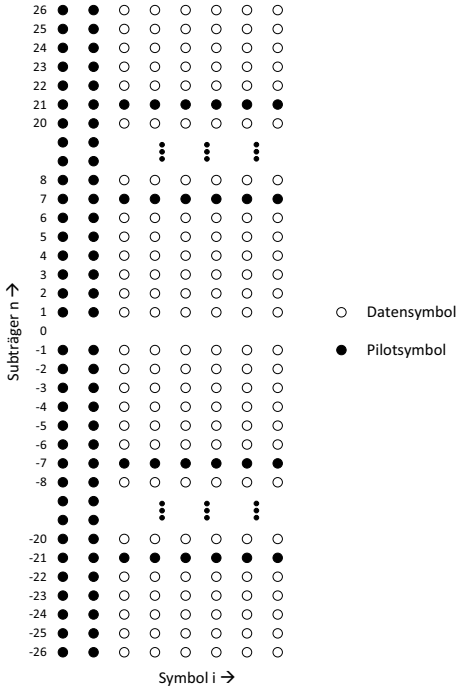


Abbildung 2.17.: Pilotenschema des IEEE 802.11a Standards. Die ersten zwei OFDM-Symbole bilden die Präambel und bestehen daher nur aus Pilottönen [46].

verwendet werden, sondern einen Referenzwert (Pilotton) senden. Dabei gibt es verschiedene Möglichkeiten die Pilottöne zu verteilen, vergleiche Abbildung 2.18.

Beim Block-Muster (Abbildung 2.18 a)) werden auf allen Trägern zu bestimmten Zeitpunkten die Pilottöne gesendet, es handelt sich im Prinzip also um eine Präambel-basierte Schätzung, vgl. Kapitel 2.1.6.1. Werden wie in Abbildung 2.18 b) auf bestimmten Frequenzen durchgehend Piloten gesendet, so nennt man das Kamm-Muster. Natürlich ist es auch möglich, die Verfahren zu kombinieren oder die Position der Pilotträger im Laufe der Zeit zu ändern, wie beispielsweise in Abbildung 2.18 c) zu sehen (scattered pilots). Wichtig ist hierbei, dass der Empfänger immer weiß, welche Träger zu welchem Zeitpunkt die Pilottöne enthalten. Weitere Informationen finden sich neben [46] auch in [39] sowie [84].

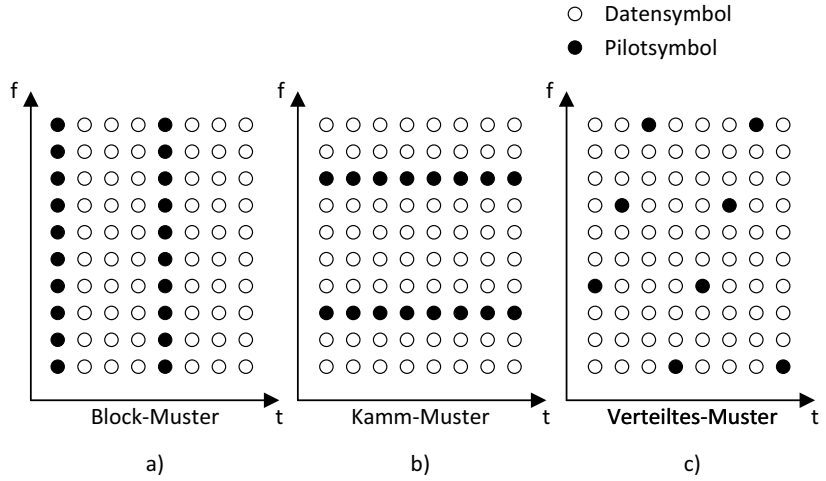


Abbildung 2.18.: Verschiedene Arten der Pilotenverteilung: Je nach Muster muss zur Kanalkorrektur später in der Zeitebene, in der Frequenzebene, oder in beiden Ebenen interpoliert werden, vgl. [84] und [46].

Kanalschätzungsverfahren nutzen oftmals Interpolationstechniken, um die Kanalschätzung zu ermöglichen bzw. zu verbessern. So ist es beispielsweise bei der Kanalschätzung mit Piloten im Kamm-Muster unvermeidbar, eine Interpolation der erhaltenen Pilotträger-Korrekturwerte über die Frequenzwerte durchzuführen um geeignete Werte für die Korrektur der Datenträger zu erhalten. Eine Schätzung durch eine Pilotenverteilung im Blockmuster kann oftmals verbessert werden, wenn in der Zeit zwischen den Pilotwerten zweier Blöcke interpoliert wird.

Somit besitzt die Verteilung der Pilottöne, wie in Abbildung 2.18 dargestellt, einen Einfluss auf die in der Kanalkorrektur notwendige Interpolation, da die Pilottonträger die Stützstellen der Interpolation bilden. Der verwendete Interpolationsalgorithmus wird durch die benötigte Güte als auch durch die zur Verfügung stehenden Rechenkapazität bestimmt. Auch das Verhältnis von Pilotträgern zu Datenträgern hat einen Einfluss auf die benötigte Güte der Interpolation.

2.1.7. Error-Vector-Magnitude (EVM)

Die Größe eines Fehlervektors (Error-Vector-Magnitude, EVM), beziehungsweise das Mittel der Größe aller Fehlervektoren, kann als Maß zur Beurteilung der Qualität eines modulierten Signals dienen, vgl. [83], [SNW+12] und [FSN+12]. Dabei gibt die EVM die Entfernung des demodulierten Wertes im Konstellationsdiagramm zu dem entsprechenden Referenzwert an. Ob es sich hierbei um den nächsten gültigen Wert handelt oder um den korrekten Wert, hängt davon ab, ob die korrekten Werte verfügbar sind. Bei entsprechender Verfügbarkeit handelt es sich um eine *data-aided* Messung, anderenfalls um eine *non data-aided* Messung.

Die Größe des Fehlervektors wird hierbei in Bezug auf einen Referenzwert angegeben. Eine Variante nutzt als Referenz den größten Vektor der verwendeten Modulationsart (EVM_M), eine Alternative nutzt die durchschnittliche Größe der Modulationsvektoren (EVM_A). Abbildung 2.19 zeigt das Beispiel eines Fehlervektors in grün, den größten Referenzvektor für eine 16-QAM Modulation rot und den Referenzvektor mit der durchschnittlichen Größe der 16-QAM Vektoren in blau.

$$EVM_M = \frac{|P_{Error}|}{|P_{Ref1}|} \cdot 100\% \tag{2.26}$$

Der (EVM_M) Wert, welchem in dieser Arbeit der Vorzug gegeben wurde, wird in Prozent angegeben und bildet sich für eine 16-QAM Modulation somit anhand der Gleichung 2.26.

	QPSK	**16-QAM**	**64-QAM**
k	1	9/5	7/3

Tabelle 2.2.: Faktor k zum Umrechnen von EVM_M in EVM_A [SNW+12].

Dabei lassen sich EVM_M Werte durch Multiplikation mit einem modulationsabhängigen Faktor k, siehe Tabelle 2.2, in EVM_A Werte überführen. Eine Schätzung der Bitfehlerrate (Bit error rate, BER) durch Umrechnung der EVM mittels Formel 2.27 ist streng genommen nur für data-aided Messungen korrekt. Jedoch wurde in [SNW+12] gezeigt, dass dies auch für non-data-aided Messungen angewendet werden kann, falls die Bitfehlerrate 10^{-2} nicht unterschreitet.

$$BER \approx \frac{(1 - L^{-1})}{\log_2(L)} \operatorname{erfc}\left(\sqrt{\frac{3\ \log_2(L)}{(L^2-1)} \frac{1}{(k\ \mathrm{EVM_M})^2\ \log_2(M)}}\right) \tag{2.27}$$

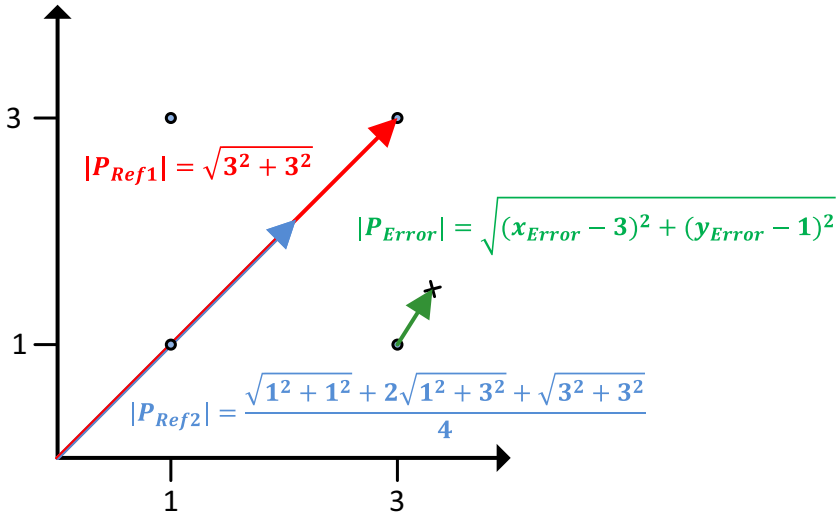


Abbildung 2.19.: Die Abbildung zeigt den ersten Quadraten eines Konstellationsdiagramms für 16-QAM. Der grüner Vektor ist ein Beispiel für einen Fehlervektor, der rote und der blau Vektor verdeutlichen die Referenzvektoren bezogen auf den maximalen Vektor (rot), als auch auf das Mittel (blau).

Führt man zur Abschätzung also eine Konvertierung der EVM_M zur Bit Error Rate (BER) durch, erhält man die Werte aus Tabelle 2.3 für eine Bitfehlerrate von 10^{-3}. Durch den Einsatz von Vorwärtsfehlerkorrektur (Forward Error Correction, FEC) der dritten Generation in modernen Kommunikationssystemen können für eine fehlerfreie Übertragung sogar Bitfehlerraten von $1,9 \cdot 10^{-}2$ bei 25% Overhead und Bitfehlerraten von $2,3 \cdot 10^{-3}$ bei einem Overhead von 7% toleriert werden, vergleiche [64].

Tabelle 2.3.: Modulationsabhängige Umsetzungstabelle für eine BER von 10^{-3} zu entsprechenden EVM_M Werten.

BER	EVM_M BPSK	EVM_M QPSK	EVM_M 16-QAM	EVM_M 64-QAM
10^{-3}	45,8%	32,5%	11,1%	4,9%

2.2. Field Programmable Gate Arrays (FPGA)

Ein Field Programmable Gate Array (FPGA) ist ein feingranular rekonfigurierbarer, integrierter Schaltkreis (Integrated circuit, IC). Dies bedeutet, dass die Funktionalität des ICs zum Zeitpunkt der Herstellung noch nicht festgelegt wird. Sie wird vom Anwender vor Ort entwickelt und anschließend als *Konfiguration* in den entsprechenden Konfigurationsspeicher des Bausteines geschrieben. Je nach Technologie lässt sich diese Konfiguration nahezu beliebig oft ändern.

Es gibt mehrere Möglichkeiten einen FPGA zu realisieren, die größten Unterscheidungsmerkmale liegen dabei im Typ des Speichers, welcher die Konfiguration des Chips hält (beispielsweise SRAM, Antifuse, Flash), sowie in der Art und Weise, wie die elementaren Zellen des FPGAs realisiert werden (beispielsweise mittels Umsetzungstabelle oder per Multiplexer). Weiterführende Informationen finden sich in [17].

Bis auf wenige Ausnahmen für spezielle Fälle haben sich in der Industrie und Forschung die auf Umsetzungstabellen (Look-Up-Table, LUT) basierte Bausteine der Firmen Xilinx und Altera durchgesetzt. Bei beiden wird dabei die Konfiguration in flüchtigem Static Random-Access Memory (SRAM) gespeichert. Sie sind schneller und bieten trotzdem mehr Logikressourcen als ihre Konkurrenz, wodurch die wenigen Nachteile, wie beispielsweise der Verlust der Konfiguration bei Trennung der Energieversorgung in Kauf genommen wird.

Da in dieser Arbeit FPGAs vom Typ Virtex-6 von Xilinx zum Einsatz kommen, beziehen sich die folgenden Erläuterungen im Detail auf Bausteine dieses Herstellers. Die grundlegenden Konzepte finden sich jedoch zumeist auch bei den Bausteinen anderen Herstellern wie beispielsweise Altera.

2.2.1. Typische Anwendungsgebiete für FPGAs

Durch die Möglichkeit, prinzipiell jede beliebige digitale Schaltung auf einen FPGA abzubilden, falls dieser über genügend Ressourcen verfügt, bieten FPGAs in vielen Bereichen eine echte Alternative zu einer Realisierung als anwendungsspezifische integrierte Schaltung (application-specific integrated circuit, ASIC). Von bestimmten Vorteilen in Nischenanwendungen abgesehen, liegt die Stärke von FPGAs in der Flexibilität, das Design jederzeit zu ändern und somit Fehlerkorrekturen oder Anpassungen ohne viel Aufwand, auch sehr spät im Entwicklungsprozess, durchführen zu können. Dies führt zu wesentlich geringeren Entwicklungszeiten und somit zu kürzeren Produkteinführungszeiten.

Des Weiteren ermöglichen FPGAs in der Regel durch niedrigere Fixkosten eine teilweise wesentlich kostengünstigere Realisierung bei geringen bis mittle-

ren Stückzahlen, im Vergleich zu einer Fertigung als ASIC. Als Nachteil wird eine Realisierung mittels eines FPGA, im Vergleich zur ASIC Lösung, durch den Overhead der Rekonfigurierbarkeit, nur mit geringeren Taktraten und mit einer schlechteren Energiebilanz zu betreiben sein. Auch ist ab einer bestimmten Stückzahl die Fertigung als ASICs kostengünstiger. Detaillierte Vergleiche zwischen der Leistungsfähigkeit einer ASIC-Implementierung und einer Realisierungen per FPGA finden sich in [51].

Laut dem Finanzbericht [115] für das vierte Quartal 2014 verteilt sich der Umsatz bei Xilinx dabei wie folgt auf den Markt:

- 49% Communications & Data Center
- 34% Industrial, Aerospace & Defense
- 15% Broadcast, Consumer & Automotive
- 2% Other

2.2.2. Allgemeine Struktur

Die Grundstruktur eines FPGAs ist in Abbildung 2.20 dargestellt. Sie ermöglicht es, viele Basiszellen flexibel miteinander zu verbinden. Die Basiszellen entsprechen dabei den *Logikzellen*, welche je nach Konfiguration beliebige logische Grundoperationen (z.B. Negation, Konjunktion, oder auch das Speichern) realisieren können. Die flexiblen Verbindungen werden dabei durch *Schaltmatrizen* gesteuert, welche die Verbindungen ermöglichen oder sperren. Natürlich sind auch Ein- und Ausgabe Elemente (IO-Pad) sowie ihre Treiber (IO-Buffer bzw. IOB) vorhanden. Modernen FPGAs ist es möglich, durch ihre komplexen konfigurierbaren Ein- und Ausgänge eine Vielzahl von Schnittstellen-Standards zu unterstützen.

Mit Hilfe dieser Basiskomponenten kann, genügend Ressourcen vorausgesetzt, jede beliebige digitale Schaltung durch ein FPGA realisiert werden. Vergleicht man den Aufbau einer komplexen Schaltung mittels solcher rekonfigurierbarer Zellen allerdings mit einem direkten Aufbau durch Transistoren welcher sich nicht rekonfigurieren lässt (z.B. Standardzellen), so benötigt die rekonfigurierbare Realisierung aufgrund der zusätzlichen Rekonfigurationslogik und den zusätzlichen Randbedingungen, naturgemäß mehr Chipfläche und Transistoren. Deshalb werden in modernen FPGAs die Logikzellen durch einige Spezialkomponenten, welche in fast jedem Design benötigt werden, aber nur sehr aufwendig durch die Basiszellen realisiert werden können, ergänzt, siehe 2.2.4.

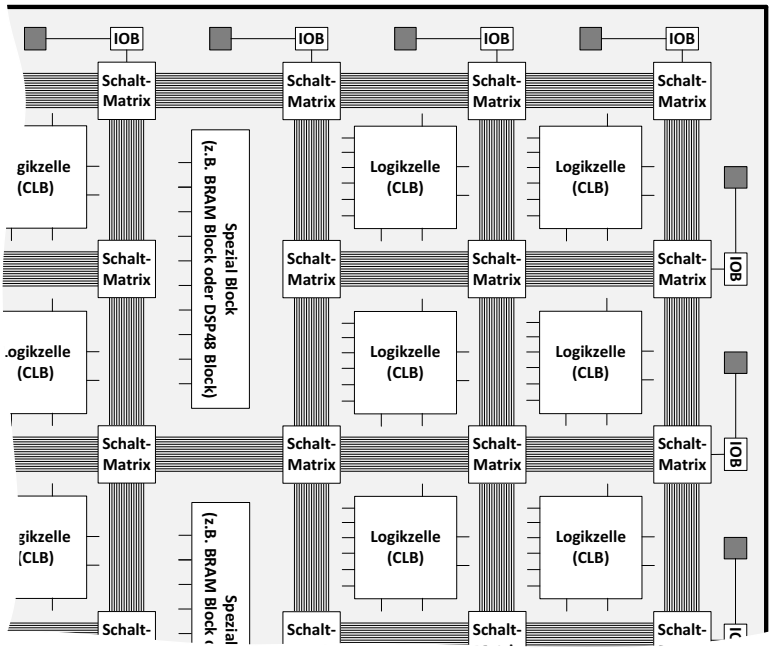


Abbildung 2.20.: Prinzipielle Struktur eines FPGAs. Die Basiskomponenten bilden die Pads und die Treiber für die Ein- und Ausgänge, die Logikzellen, sowie ein Verbindungsnetzwerk welches durch Schalt-Matrizen gesteuert wird. Moderne Bausteine verfügen außerdem über Spezialkomponenten wie Speicher oder Multiplizier-Einheiten.

2.2.3. FPGA Basiskomponenten

Die wichtigste Komponente innerhalb eines FPGA ist eine Basiszelle, auch Logikzelle genannt. Xilinx nennt diese Zellen *Configurable Logic Block* (CLB).

Bei der Virtex-6 Generation von Xilinx wird ein CLB in 2 SLICE-Komponenten unterteilt. Ein SLICE wiederum gibt es in 2 Ausführungen, SLICEM und die eingeschränkten SLICEL. Ein SLICEM wird in Abbildung 2.21 gezeigt. Links im Bild sind 4 Logikelemente zu sehen, danach folgt eine Carry-Chain, um benachbarte SLICE Komponenten optimiert zu verschalten und dadurch effizientere Rechen-Strukturen aufbauen zu können. Schließlich enthält ein CLB noch mehrere Speicherelemente, um beispielsweise ein D-Flip-Flop zu realisieren.

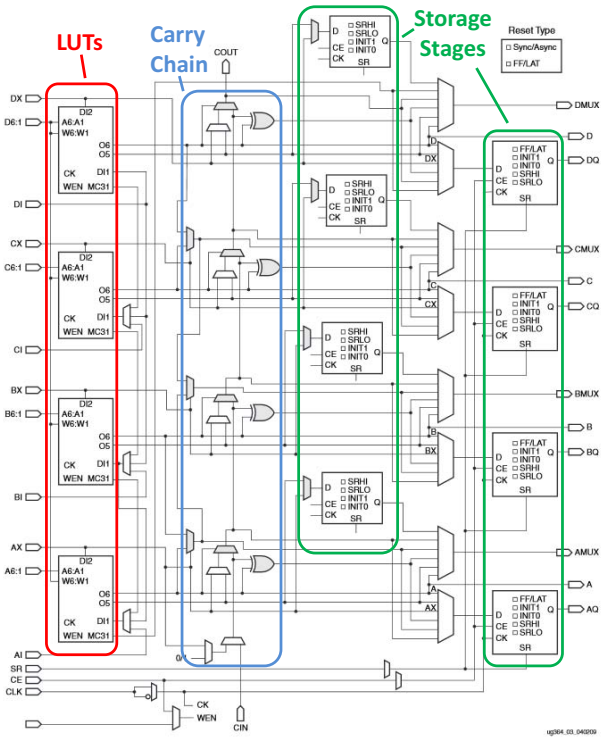
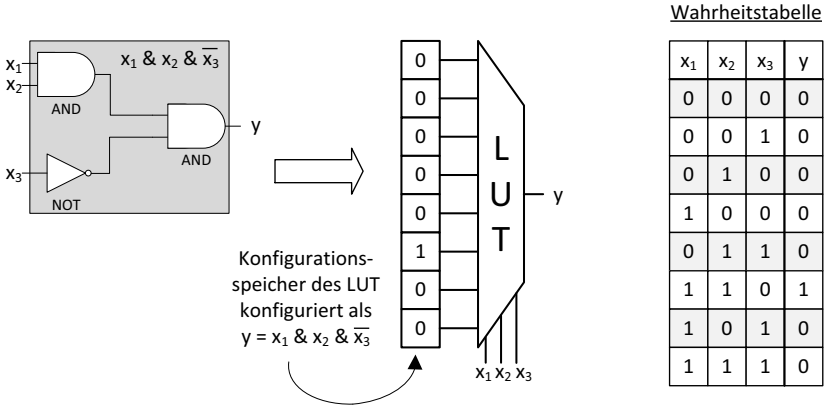


Abbildung 2.21.: SLICEM eines Virtex-6 FPGAs. Von links nach rechts finden sich 4 LUTs (auch als Speicher konfigurierbar), eine Carry-Chain und insgesamt 8 speichernde Elemente [106].

Alle Elemente innerhalb eines CLBs können durch Multiplexer oder über die Verwendung der entsprechenden Ein- und Ausgänge in den Signalweg integriert werden. Einige der Elemente stellen mehrere Funktionen zur Verfügung. So können die Logikzellen der SLICEM Komponenten (Funktionsgeneratoren genannt) entweder als *Umsetzungstabelle* (Look-Up-Table, LUT) zur Realisierung von Logikfunktionen, als Speicher oder als 32-Bit Schieberegister konfiguriert werden. Detailliertere Informationen finden sich im entsprechenden User Guide [106].

Um nachzuvollziehen, wie es einem FPGA möglich ist, mit Hilfe dieser CLBs flexibel komplexe Schaltkreise zu realisieren, ist es notwendig, das kleinste Kernelement, eine Umsetzungstabelle, zu verstehen. Diese setzt sich aus einem Konfigurationsspeicher sowie einem Multiplexer zusammen, dessen Steuersignale die



Wahrheitstabelle

x_1	x_2	x_3	y
0	0	0	0
0	0	1	0
0	1	0	0
1	0	0	0
0	1	1	0
1	1	0	1
1	0	1	0
1	1	1	0

Abbildung 2.22.: Realisierung der Schaltfunktion $y = x_1 \,\&\, x_2 \,\&\, \overline{x_3}$ durch eine Umsetzungstabelle (LUT). Die Select Eingänge des Multiplexers stellen die Eingänge des LUT dar.

Eingänge des LUTs bilden. Die Eingänge des Multiplexers sind wiederum so mit dem Konfigurationsspeicher verbunden, dass durch die Steuersignale und damit durch die Eingänge des LUTs bestimmt wird, welche Konfigurationsspeicherzelle durch ihren Inhalt den Wert am Ausgang des Multiplexers vorgibt.

Da sich der Konfigurationsspeicher eines LUT beliebig mit Werten füllen lässt, lassen sich damit auch beliebige Funktionen realisieren. Die Komplexität eines einzelnen LUT ist hierbei durch die Anzahl an Ein- und Ausgängen bestimmt. Die LUTs moderner Xilinx FPGAs besitzen 6 Eingänge und 2 Ausgänge. Zur einfacheren Darstellung zeigt Abbildung 2.22 das Prinzip eines LUT mit nur 3 Eingängen und einem Ausgang, anhand der Realisierung eines 3-fach AND-Gatters mit zwei regulären und einem negierten Eingang.

Neben der flexiblen Realisierung von Logikfunktionen ist es notwendig, diese ebenso flexibel verdrahten zu können. In einem FPGA sind alle Elemente bereits zu sogenannten Schaltmatrizen vorverdrahtet. Die Schaltmatrizen ermöglichen ein flexibles Verbinden ihrer Ein- bzw. Ausgänge. Das Prinzip ist in Abbildung 2.23 verdeutlicht. Um einen Eingang mit einem Ausgang zu verbinden muss die Speicherzelle des Konfigurationsspeichers für den entsprechenden Transistor so gesetzt werden, dass dieser leitet. Alle Transistoren, welche den Eingang auf einen Ausgang schalten, der nicht erwünscht ist, müssen entsprechend in einen nicht leitenden Zustand versetzt werden.

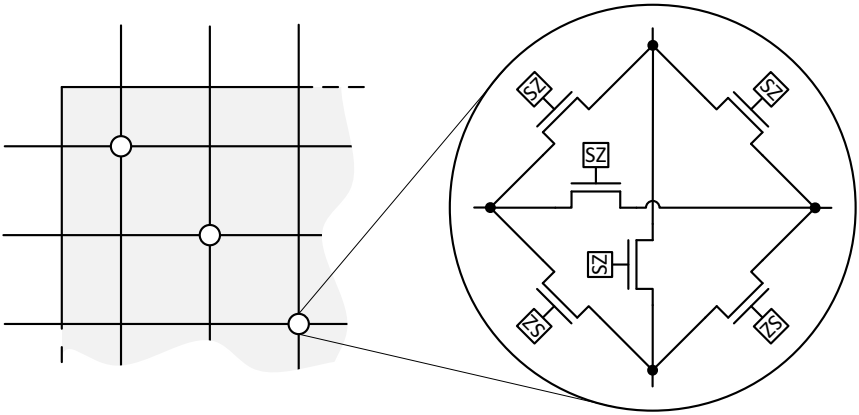


Abbildung 2.23.: Das Prinzip einer Schaltmatrix ermöglicht ein flexibles Herstellen von Verbindungen innerhalb des FPGAs. Dabei bestimmt der Inhalt der Speicherzellen (SZ) des Konfigurationsspeichers den Zustand einer möglichen Verbindung.

2.2.4. Spezielle Elemente moderner FPGAs

Die erste wichtige Spezialkomponente, deren Funktionalität sich nur aufwendig mit einer hohen Anzahl an CLBs realisieren lassen würde, ist Speicher. Deshalb gibt es spezielle, fest implementierte Speicher-Elemente, von Xilinx *Block-RAM (BRAM)* genannt. Ein einzelner BRAM eines Virtex-6 FPGAs bietet bis zu 36 Kilobits an Speicher sowie zwei Schnittstellen auf diesen Speicher, welche gleichzeitig und unabhängig voneinander betrieben werden können. Die größten Virtex-6 FPGAs verfügen über 1064 solcher BRAMs womit sich über 38 000 Kilobit Speicher effizient realisieren lassen. Mehr Informationen finden sich in [112].

Die zweite wichtige Funktion, für welche spezielle Einheiten existieren, ist die Multiplikation. Während frühe Generationen dieser speziellen Einheiten lediglich eine Multiplikation unterstützten, wurden nach und nach immer mehr zusätzliche Funktionalitäten integriert. So bieten die *DSP48E1 Slice* genannten Einheiten der Virtex-6 FPGAs nicht nur einen 25-Bit x 18-Bit Multiplizierer, sondern ebenso vorangehende Additionen, nachfolgende Additionen, spezielle Schiebeoperationen, Vergleichsoperationen, und vieles mehr. Zusätzlich sind auch Register direkt in den DSP48E1 Slices enthalten, um für den Einsatz von Pipelining nicht auf CLBs angewiesen zu sein, vergleiche [101].

Um die enorme Rechenleistung moderner FPGAs voll ausnutzen zu können, ist es notwendig, die zu verarbeitenden Daten rechtzeitig in den Chip und die Ergebnisse entsprechend aus dem Chip heraus zu transportieren. Dazu besitzen viele der größeren FPGAs serielle Hochgeschwindigkeits-Transceiver zur Erhöhung der IO-Bandbreite. Bei der Virtex-6 Familie unterscheidet man dabei zwischen den *GTX Low-Power Transceivers* und den *GTH High-Speed Transceivers*. Je nach Ausführung unterstützt ein Transceiver eine Vielzahl von Leitungskodes (z.B. 8B/10B) und andere Hilfsmittel (z.B. Präemphase) um eine stabile Datenübertragung zu erreichen. Dabei werden pro Transceiver nominale Datenraten von 6,6 Gb/s (GTX) beziehungsweise 11,18 Gb/s (GTH) erreicht, vergleiche [103] sowie [102].

Neben den bereits vorgestellten Spezialkomponenten kommen auch zur Erzeugung, Veränderung und Verdrahtung des Taktes spezielle, eigene Ressourcen zum Einsatz. So existieren nicht nur spezielle PLL-Primitive und Treiber für das Taktsignal, auch dessen Verteilung geschieht in der Regel über eigens dafür vorgesehene Routing-Ressourcen.

Je nach Familie und Generation befinden sich auch einzelne dedizierte Schaltungen bis hin zu ganzen Prozessoren auf FPGAs, um auch hier effiziente Realisierungen für häufig benötigte Funktionen zu bieten. Ein Beispiel sind die PCI-Express Schnittstellen der Virtex-6 FPGAs oder die PowerPC-Kerne, welche in einigen Virtex-5 FPGAs verfügbar sind.

Tabelle 2.4.: Kerneigenschaften der Virtex-6 FPGA Familie anhand ausgewählter Bausteine im Vergleich.

	Baustein	**XC6VLX75T**	**XC6VHX380T**	**XC6VLX760**
Logik Ressourcen	Logikzellen	74 496	382 464	758 784
	Slices	11 640	59 760	118 560
	CLB-Flip-Flops	93 120	478 080	948 480
Speicher Ressourcen	Block-RAM (#)	156	768	720
	Block-RAM (Kb)	5 616	27 648	25 920
Ein-/Ausgabe Ressourcen	max. Asymmetrisch	360	720	1 200
	max. Symmetrisch	180	360	600
Weitere spezielle Ressourcen	DSP48E1 Slices	288	864	864
	PCIe Blocks	1	4	-
	Ethernet MACs	4	4	-
	GTX Transceiver	12	48	-
	GTH Transceiver	-	24	-
Konfiguration	Konfig.-Speicher	26,3 Mb	184,9 Mb	119,8 Mb

Die Kerneigenschaften der Virtex-6 Familie sind in Tabelle 2.4 aufbereitet. Es werden verschiedene Typen von Virtex-6 FPGAs verglichen, grau hinterlegte Bausteine wurden in dieser Arbeit verwendet. Für Details zu den einzelnen Komponenten sei auf die entsprechenden Dokumente des Herstellers verwiesen [105].

2.2.5. FPGA Entwurfsablauf

Zu Beginn des Entwurfsablaufes eines jeden Hardware Designs sollte, je nach Komplexität der zu implementierenden Schaltung beziehungsweise des zu implementierenden Algorithmus, zuerst eine Hochsprache wie C++ oder Matlab verwendet werden, um schnell und effizient die Funktionsweise zu evaluieren, vergleiche Abbildung 2.24 Punkt 1. In vielen Fällen sind in den Bibliotheken dieser Programmiersprachen bereits Realisierungen des Algorithmus vorhanden. In einigen Fällen kann es jedoch auch Sinn machen, den Algorithmus noch einmal selbst zu programmieren, um dessen Funktionsweise zu verstehen. Dabei ist es möglich, diesen in einer Art und Weise zu implementieren, welche der späteren Hardware Realisierung besonders nahe kommt. Ein Beispiel hierfür wäre die Verwendung desselben Zahlenformats. Die Implementierung auf einer hohen Abstraktionsebene soll nicht nur sicherstellen, dass man den Algorithmus verstanden hat und dieser auch das gewünschte Ergebnis liefert, sie soll im Folgenden auch als Referenzmodell verwendet werden.

In Schritt 2 wird nun die Verhaltensbeschreibung der Schaltung erzeugt. Auch wenn es starke Bemühungen gibt, diese Beschreibung aus der Referenzimplementierung von Punkt 1 zu gewinnen, wird der Code der *Hardwarebeschreibungssprache* (Hardware Description Language, HDL) in den meisten Fällen noch von Hand erstellt und optimiert. Mittels einer Simulationen auf Registertransferebene (Register Transfer Level, RTL) kann die Beschreibung überprüft und nach Bedarf korrigiert oder optimiert werden. Diese Regelschleife stellt Schritt 3 dar.

Der abschließende Test der Schaltung geschieht in Schritt 4. Hierzu bietet sich eine Co-Simulation an. Dabei können dieselben Eingangsdaten wie für die Referenzmodellierung verwendet werden, die Simulation des Algorithmus findet allerdings auf RTL-Ebene statt. Das Ergebnis kann dann in der Entwicklungsumgebung der Hochsprache mit der Referenzimplementierung verglichen werden.

Unterschiedliche Ergebnisse können von Fehlern in der Implementierung stammen, oder aber von den unterschiedlichen Implementierungsarten (beispielsweise durch unterschiedlichen Zahlendarstellungen) und müssen nun auf ein Maß gebracht werden, welches für die Zielanwendung hinreichend nahe an der Referenz liegt. Durch eine Co-Simulation können, im Gegensatz zur reinen Simulation auf RTL-Ebene, relativ leicht und automatisiert eine Vielzahl von Eingangsda-

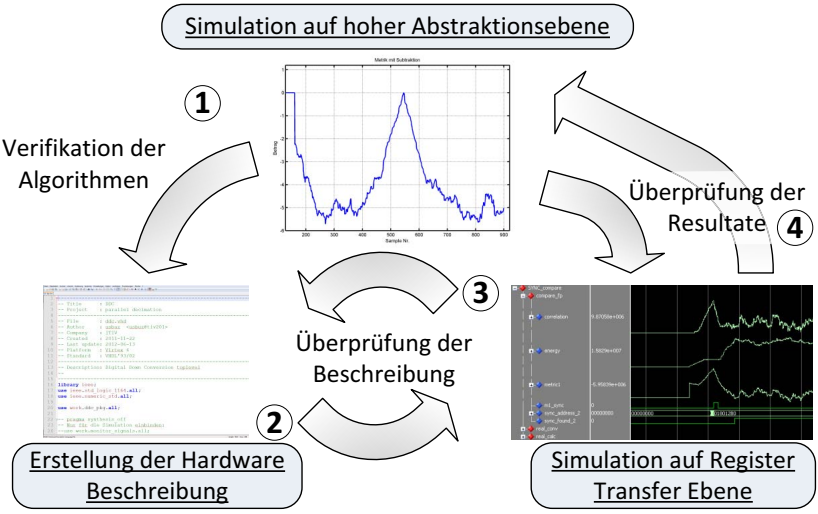


Abbildung 2.24.: Typischer Ablauf der Erstellung einer Hardwarebeschreibung: Zuerst wird die allgemeine Funktionalität durch eine Referenzimplementierung auf hoher Abstraktionsebene überprüft und eine Simulationsumgebung geschaffen. Dann wird die Beschreibung erstellt und fortwährend durch Simulationen auf RTL-Ebene deren Verhalten verifiziert. Im letzten Schritt wird mittels einer Co-Simulation die Funktionalität der Hardwarebeschreibung mit der Referenzimplementierung verglichen.

ten und Anwendungsfällen getestet, sowie die Ergebnisse anschaulich dargestellt werden.

Liegt die Hardwarebeschreibung nun fehlerfrei vor, kann durch die Hardwaresynthese eine Konfiguration für den FPGA erstellt werden. Die Konfiguration bzw. die Datei, welche zum Konfigurieren des FPGAs genutzt wird, heißt *Bitstrom*. Um von der Hardwarebeschreibung auf einen gültigen Bitstrom zu kommen werden in der *Hardwaresynthese* vier Schritte durchlaufen, vergleiche Abbildung 2.25.

Im ersten Schritt, der eigentlichen Synthese, wird aus der Verhaltensbeschreibung eine Netzliste generiert, in welcher alle Funktionalität auf eine Kombination entsprechender Logikgatter und deren Verbindung reduziert wurde. Die Schaltung liegt nun in einer RTL-Beschreibung vor. Diese Gatter der Netzliste werden nun im zweiten Schritt, der *Technologieabbildung*, auf die entsprechenden Primitive der

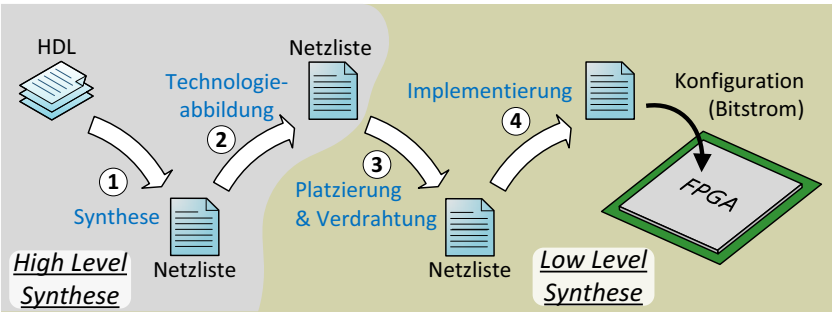


Abbildung 2.25.: Grobe Einteilung der Hardwaresynthese in die durchzuführenden Schritte.

Zielarchitektur abgebildet. Im Falle eines Xilinx-FPGAs also auf Elemente wie z.B. LUTs, BRAM oder Flip Flops. Als Resultat erhält man eine Netzliste, welche sich aus diesen Primitiven und ihrer Verdrahtung zusammensetzt.

Werden die vorherigen Schritte als High-Level Synthese bezeichnet, entsprechen die nachfolgenden Schritte der Low-Level Synthese. Diese besteht zum Großteil aus Schritt 3, der *Platzierung und Verdrahtung*. Wie der Name schon nahe legt, werden hier die einzelnen Primitive der Netzliste nun auf konkrete Primitive im FPGA abgebildet (platziert). Anschließend wird eine Verbindung dieser FPGA-Primitive mit Hilfe der Schaltmatrizen gebildet (Verdrahtung).

Im letzten Schritt wird aus der platzierten und verdrahteten Netzliste der Bitstrom für den FPGA erzeugt. Dieser kann in den Konfigurationsspeicher geladen werden, womit die entsprechenden Primitive und die dazugehörigen Verdrahtungen aktiv werden und der Baustein die entwickelte Funktionalität realisiert.

Während all dieser Schritte wird ständig versucht, Vorgaben zur Taktzyklendauer einzuhalten und den Ressourcenverbrauch zu minimieren. Dabei können verschiedene Optimierungstechniken wie beispielsweise *Retiming* [55] oder Register Duplikation [79] verwendet werden, um eine Optimierung in Richtung geringe Taktzyklendauer oder geringer Ressourcenverbrauch zu favorisieren. Ebenso ist es möglich, einen niedrigeren Energieverbrauch als Ziel festzulegen. Nähere Informationen zu diesen und andere Optimierungstechniken finden sich in [107] sowie in [36].

2.2.6. Dynamische Partielle Rekonfiguration

FPGAs und andere rekonfigurierbare Architekturen bieten die Möglichkeit, durch das Einspielen einer neuen Konfiguration ihre Funktionalität jederzeit zu ändern. In der Regel wird hierzu das System angehalten, der Inhalt des Konfigurationsspeichers wird vollständig überschrieben, um anschließend das System wieder in Betrieb zu nehmen.

Die *partielle Rekonfiguration* beschreibt die Technik, nur einen Teil des Konfigurationsspeichers neu zu beschreiben, der übrige Speicherbereich behält seine vorherige Konfiguration bei. Wird das System zur Durchführung der partiellen Rekonfiguration gestoppt und anschließend wieder gestartet, spricht man von statischer partieller Rekonfiguration. Geschieht die Rekonfiguration im laufenden Betrieb, spricht man von *dynamischer partieller Rekonfiguration*, teils auch aktive partielle Rekonfiguration genannt.

Durch die Möglichkeit Funktionalität auszutauschen, während der Rest des Chips aktiv bleibt, lassen sich mit Hilfe dieser Technik neue Anwendungsfelder erschließen. So können beispielsweise Hardwarebeschleuniger, welche nicht zeitgleich benötigt werden, erst bei Bedarf nacheinander durch die selben Ressourcen realisiert werden. Durch diese gemeinsame Verwendung von Ressourcen lassen sich für das Gesamtdesign insgesamt Ressourcen und somit Kosten und Energie sparen.

Für FPGAs des Herstellers Xilinx gibt es traditionell zwei verschiedene Entwurfsabläufe zur Erstellung partieller Konfigurationsbitströme, die modulbasierte Rekonfiguration sowie die differenzenbasierte Rekonfiguration. Ersteres verfolgt das Ziel, Module, also ganze Teilbereiche eines Designs, zu ersetzen. Die Erstellung der partiellen Bitströme geht dabei auf eine komplette Realisierung des neuen Schaltungsteils zurück. Dabei wird jedoch der Entwurfsraum für das neue Design auf den Bereich des Chips eingeschränkt, den es abzuändern gilt. Der erste Entwurfsfluss hierfür hieß Partial Mask, welcher bald von dem Early Access Partial Reconfiguration Entwurfsfluss abgelöst wurde. Mittlerweile sind beide veraltet, der aktuelle Entwurfsfluss existiert seit der Version 12.1 der Entwicklungswerkzeuge und basiert auf den Einsatz von Partitionen.

Im Gegensatz zur modulbasierten Rekonfiguration soll die differenzenbasierte Rekonfiguration keine ganzen Funktionalitäten austauschen, sondern lediglich die Konfiguration einzelner Elemente ändern. So können zum Beispiel auf einfache Weise die Funktionen von LUTs abgeändert werden oder Verbindungen umverdrahtet werden. Dazu wird normalerweise ein Design von Hand im FPGA Editor geändert und die Differenzen zum originalen Design in einem partiellen Bitstrom gespeichert.

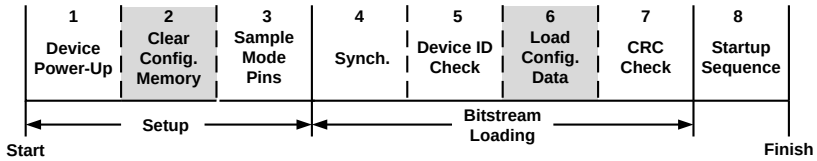


Abbildung 2.26.: Konfigurationsablauf für Virtex-6 FPGAs [110]. Die dominierenden Phasen sind grau hinterlegt.

Details zu den Vorgehensweisen und zur dynamischen partiellen Rekonfiguration allgemein finden sich in [97], sowie in [108] und [96].

2.2.7. Systemstart eingebetteter Systeme auf FPGA-Basis

Durch die Verwendung von flüchtigem Speicher als Konfigurationsspeicher ergeben sich einige Besonderheiten beim Start von Systemen auf FPGA-Basis. Diese Besonderheiten resultieren aus der Startupsequenz eines FPGAs. Für Virtex-6 FPGAs des Herstellers Xilinx ist diese in Abbildung 2.26 gegeben, sie ist für alle modernen FPGAs dieses Herstellers gleich. Die Sequenz kann in 2 Schritte geteilt werden, den *Setup*-Schritt, welcher den FPGA initialisiert (Setup) und das *Laden des Bitstroms* (Bitstream Loading).

Die dominierende Phase des Setup-Schrittes ist das *Löschen des Konfigurationsspeichers*. Die Datenblätter sprechen für die hierfür benötigte Zeit von der *Power-On-Reset* Zeit T_{POR}, welche bausteinabhängig ist und für aktuelle Virtex-6 FPGAs 55 Millisekunden beträgt. Hierbei handelt es sich allerdings um Worst-Case Angaben, also um Angaben für den schlimmsten anzunehmenden Fall, vgl. [111] und [100].

Die Dauer des zweiten Teils der Startupsequenz wird vom Laden der Konfigurationsdaten bestimmt und ist zum einen abhängig von der verwendeten Konfigurationsschnittstelle, dem verwendeten Konfigurationsmodus und des verwendeten Konfigurationstaktes, zum anderen ist natürlich auch die Menge der zu übertragenden Konfigurationsdaten ausschlaggebend. Somit lässt sich die Konfigurationsdauer eines FPGAs ausreichend genau durch Formel 2.28 schätzen.

$$T_{config} = T_{POR} + T_{Bitstream\_loading} = T_{POR} + \frac{S_{Bitstream}}{V_{config}} \tag{2.28}$$

Tabelle 2.5.: Worst-Case Geschwindigkeiten der unterschiedlichen Virtex-6 Konfigurationsschnittstellen

FPGA Familie	Konfig. Modus	Schnitt-stelle	Busbreite	Max. Taktrate	Konfigurations-Geschwindigkeit V_{config}
Virtex-6	Master	SPI	1	47,25 MHz	47,25 Mb/s
		Serial	1	47,25 MHz	47,25 Mb/s
		BPI	8	45,00 MHz	360,00 Mb/s
		BPI	16	45,00 MHz	720,00 Mb/s
		Select Map	8	45,00 MHz	360,00 Mb/s
		Select Map	16	45,00 MHz	720,00 Mb/s
	Slave	Seriell	1	100,00 MHz	100,00 Mb/s
		Select Map	8	100,00 MHz	800,00 Mb/s
		Select Map	16	100,00 MHz	1600,00 Mb/s
		Select Map	32	100,00 MHz	3200,00 Mb/s

2.2.7.1. Konfigurationsdauer moderner Virtex-6 FPGAs

Die verschiedenen Möglichkeiten einen Xilinx FPGA zu konfigurieren lässt sich in die Master- und die Slave-Modi unterteilen. Master-Modus bedeutet, dass der FPGA die Rolle des Busmasters während der Kommunikation übernimmt und somit den Takt und evtl. benötigte Steuersignale vorgibt. Im Slave Modus werden der Takt und die Steuersignale von einer externen Logik, beispielsweise einem Mikrocontroller, einem *Complex Programmable Logic Device* (CPLD) oder einem speziell dafür gefertigten Speicher von Xilinx, *Platform Flash* genannt, vorgegeben.

Prinzipiell lässt sich sagen, dass die Slave-Modi schneller sind. Der Grund hierfür liegt im Konfigurationstakt, welcher starken Schwankungen unterliegt, falls er vom FPGA mit Hilfe eines internen Oszillators erzeugt wird. So besitzt der generierte Takt bei Virtex-6 FPGAs beispielsweise eine Abweichung von bis zu $\pm$55%. Somit kann es passieren, dass man weniger als ein Viertel der möglichen Konfigurationsfrequenz erhält.

Eine weitere Kategorie, in welche sich Konfigurationsarten einteilen lassen, sind serielle und parallele Konfigurationsschnittstellen. Obwohl parallele Konfigurationsarten oftmals einen leicht geringeren maximalen Konfigurationstakt bieten, lässt sich meist, durch die bis zu 32-fache parallele Übertragung, ein höherer Durchsatz erzielen. In vielen Bereichen und vor allem in eingebetteten Systemen bildet die maximale Konfigurationsdauer und damit die maximale Geschwindigkeit der Konfigurationsschnittstelle aber nicht das entscheidende Kriterium zur Wahl der Konfigurationsmethode, auch die Kosten spielen eine erhebliche Rolle.

Tabelle 2.6.: Worst-Case Konfigurationsdauer T_{config} für Master SPI und Master BPI in 8-facher sowie in 16-facher Ausführung.

Familie	Baustein	T_{POR}	Bitstrom-Größe in Bit	T_{config} in ms SPIx1	BPIx8	BPIx16
Virtex-6	LX75T	55 ms	26.239.328	938,48	165,44	110,22
	LX130T	55 ms	43.719.776	1.527,05	239,01	147,01
	LX195T	55 ms	61.552.736	2.127,49	314,07	184,54
	LX240T	55 ms	73.859.552	2.541,86	365,86	210,43
	LX365T	55 ms	96.067.808	3.289,61	459,33	257,17
	LX550T	55 ms	144.092.384	4.906,60	661,45	358,23
	LX760	55 ms	184.823.072	6.278,00	832,88	443,94

Zur Bestimmung dieser Kosten sind allerdings nicht nur die Kosten des Speicher-Bausteins zu berücksichtigen. Auch Kosten in Form von benötigten IOs des FPGAs sowie in Form der Komplexität des Gesamtsystems, beispielsweise durch eine komplexere Leiterplatte, steigen oft mit schnelleren Konfigurationsarten. Deshalb kommen in eingebetteten Systemen, in denen die Kosten eine enorme Rolle spielen, normalerweise nur langsame Standardschnittstellen in Frage. Beispiele für solche Schnittstellen sind das *Serial Peripheral Interface* (SPI) sowie das *Byte Peripheral Interface* (BPI).

Mit den Angaben zur Größe der Konfigurationsströme aus den entsprechenden Datenblättern [110][109] lässt sich mit den Werten aus Tabelle 2.5 die Worst-Case Konfigurationsdauer nach Formel 2.28 berechnen. Tabelle 2.6 zeigt diese Dauer für eine Auswahl an Virtex-6 FPGAs und für die zur Verfügung stehenden Varianten der SPI- und BPI-Schnittstellen. Man beachte, dass es sich hierbei um die Gesamtkonfiguration handelt, das heißt dass T_{POR} bereits berücksichtigt wurde.

Anhand der errechneten Worst-Case Konfigurationszeiten sieht man leicht, dass eine schnelle Konfiguration mit der günstigen SPI-Schnittstelle für Virtex-6 FPGAs nicht möglich ist. So kann lediglich das kleinste Virtex-6 Modell Zeiten unter einer Sekunde garantieren. Man muss bedenken, dass es sich bei der Konfigurationszeit jedoch um FPGA-spezifische Zeit handelt und dass für den gesamten Start eines eingebetteten Systems zusätzliche Zeit, beispielsweise zum Starten eines Betriebssystems benötigt wird. Entsprechend ist ein schneller Systemstart für FPGA-basierte Systeme nicht ohne Weiteres möglich. Wird dennoch ein schneller Systemstart benötigt, kann *Fast Startup* eine Lösung darstellen.

2.2.7.2. Fast Startup für Xilinx-FPGAs

Fast Startup dient dem beschleunigten Start eingebetteter, FPGA-basierter Systeme und wurde parallel zu dieser Arbeit in Zusammenarbeit mit der Firma Xilinx entwickelt. Während sich die nachfolgenden Informationen konkret auf FPGAs der Firma Xilinx beziehen und alle Tests bisher lediglich mit solchen FPGAs durchgeführt wurden, sollte sich das Prinzip auf alle FPGAs anwenden lassen, die eine dynamische partielle Rekonfiguration unterstützen.

Da man die Power-On-Reset Zeit aus Formel 2.28 nicht beeinflussen kann und auch die Wahl einer schnelleren Konfigurationsschnittstelle oft nicht möglich oder ausreichend ist, besteht das Fast Startup Konzept darin, die Anzahl an Konfigurationsbits zu verringern, indem ein sequentieller Konfigurationsablauf verwendet wird. Hierbei ist es notwendig, das Gesamtsystem des FPGAs in 2 Partitionen aufteilen zu können. Eine zeitkritische Partition P-I beinhaltet dabei alle Komponenten und Funktionalitäten, welche schnellstmöglich zur Verfügung stehen müssen. Die Partition P-II enthält den Rest des Systems, also alles, was nicht notwendigerweise sofort benötigt wird.

Fast Startup unterscheidet sich von einer herkömmlichen Konfiguration durch die Tatsache, dass zu Beginn nicht der vollständige Baustein konfiguriert wird, sondern im ersten Schritt nur Partition P-I in einem möglichst kleinen Teilbereich konfiguriert wird. Dies realisiert die zu Beginn notwendigen Funktionalität. Anschließend, sobald die zeitkritische Funktionalität gegeben ist, kann in einem zweiten Schritt der Rest des FPGA-Designs mit den weniger zeitkritischen Komponenten durch dynamisch partielle Rekonfiguration in Betrieb genommen werden. Abbildung 2.27 veranschaulicht diesen sequentiellen Vorgang.

Eine Besonderheit ist hier dadurch gegeben, dass Fast Startup die erste Technik ist, welche als Initialbitstrom einen partiellen Bitstrom verwendet, alle anderen Vorgehensweisen nutzen partielle (Re-)Konfiguration lediglich, um die Konfiguration eines bereits vollständig konfigurierten FPGAs abzuändern. Für Fast Startup ist es initial jedoch nicht notwendig den vollständigen FPGA zu konfigurieren, da das Löschen des Konfigurationsspeichers zu Beginn des Systemstarts schon für eine Initialisierung aller Konfigurationszellen sorgt. Detailliertere Informationen zu diesem Konzept lassen sich in den Veröffentlichungen [HMS+10] und [MHB+11], als auch im Patent [SHNS+11] über diese Technik nachlesen.

Während man für die Erstellung des partiellen Bistroms zur Konfiguration von P-II die Herstellerprogramme für dynamische partielle Rekonfiguration verwenden kann, gibt es vom Hersteller keinen Entwurfsfluss, welcher für die Erstellung eines partiellen Bitstroms zur initialen Konfiguration von P-I geeignet ist. Daher wurde ein entsprechender Entwurfsfluss in [MNH+11a] und [MNS+11] entwickelt. Dabei wird nicht nur der Entwurfsfluss für Fast Startup im Detail erklärt, es wird auch eine Möglichkeit aufgezeigt, wie sich korrekte partielle Bitströme für

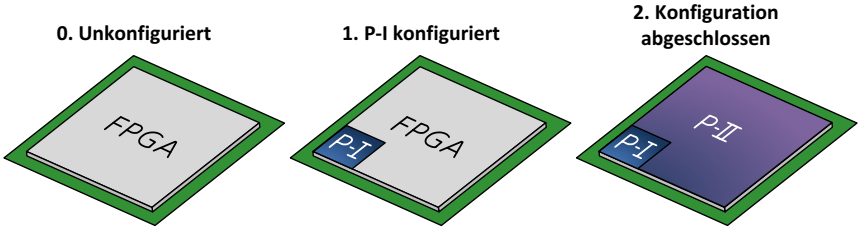


Abbildung 2.27.: Prinzipieller Ablauf der sequentiellen Fast-Startup Technik: Ausgehend von dem unkonfigurierten FPGA wird zuerst die zeitkritische Partition P-I konfiguriert. Während dieses Initialdesign dann bereits arbeitet, wird parallel die restliche Funktionalität mit dynamischer partieller Rekonfiguration auf den FPGA gebracht.

die Spartan-6 Architektur, der Low-Cost FPGA-Familie von Xilinx, erstellen lassen. Dies war notwendig, da die Herstellerprogramme für dynamische partielle Rekonfiguration diese Architektur nicht unterstützten.

2.2.7.3. Fast Software Startup für FPGAs

Die Konfigurationsdauer eines FPGAs zu reduzieren ist ein wichtiger Schritt zur Beschleunigung des Systemstarts eingebetteter, FPGA-basierter Systeme. Um ein solches System schnellstmöglich zu starten muss jedoch auch der Start der Software optimiert werden, vergleiche Abbildung 2.28. Geht man von einem kleinen, einfachen Betriebssystem aus, wie das in eingebetteten Systemen meist der Fall ist, wird ein großer Teil des Startvorgangs der Software dazu benötigt, den auszuführenden Programmcode von einem externen, nichtflüchtigen Speicher aus in den Arbeitsspeicher des Systems zu kopieren. Das direkte Ausführen der Software aus dem nichtflüchtigen, externen Speicher ist im Allgemeinen zu langsam und kommt nur in Ausnahmefällen zum Einsatz, siehe [11].

Ein Konzept, ähnlich dem von Fast Startup für die Konfiguration, wurde für den Start der Software entwickelt. Gerade bei der Verwendung von Fast Startup zur Konfiguration der Hardware wird schnell klar, dass sich auch die Software in zwei Klassen trennen lässt. So ist es nicht nötig, Programmcode in den Arbeitsspeicher zu laden, welcher Hardware benötigt, die noch gar nicht verfügbar ist. Entsprechend kann auch das Kopieren des Programmcodes in zwei Phasen unterteilt werden. Eine Darstellung eines Fast Startup für Hardware und Software ist in Abbildung 2.29 gegeben.

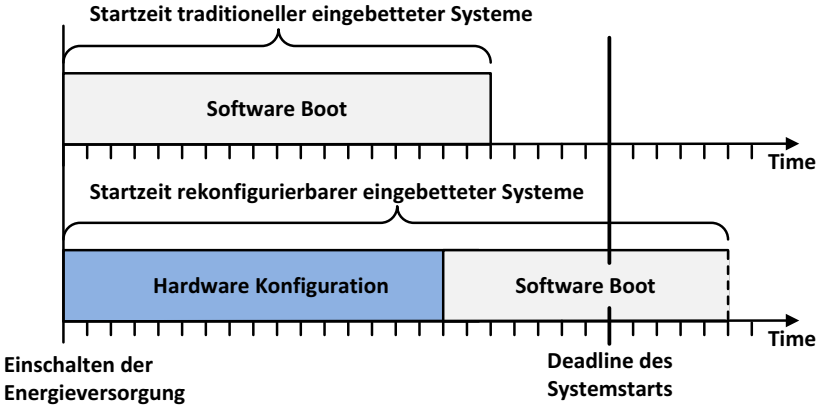


Abbildung 2.28.: Vergleich des Starts traditioneller und rekonfigurierbarer Systeme: Durch die zusätzliche Konfigurationsdauer ist es für rekonfigurierbare Systeme im Allgemeinen schwerer, Fristen für den Systemstart einzuhalten.

Das Konzept eines zweigeteilten Kopierens des Programmcodes bringt im Vergleich zu einem normalen Startvorgang neue Möglichkeiten, aber auch Nebenbedingungen mit sich. Es entstehen neue Aspekte in Bezug auf die Speicherverteilung eines entsprechenden Systems. So kann die Verwendung einer hybriden Speicherarchitektur entscheidende Vorteile mit sich bringen, vgl. [MNH+13].

Eine solche Speicherarchitektur verzichtet auf einen ressourcenintensiven Speichercontroller in der zeitkritischen Hardwarepartition, die zeitkritische Software wird aus schnellem on-Chip BRAM Speicher ausgeführt und kann so sehr früh, direkt nach der initialen Konfiguration, gestartet werden. Die weniger kritische Software wird am Ende der gesamten Konfiguration über den in P-II verfügbaren Controller in den externen Speicher geschrieben und von dort ausgeführt. Die drei Schritte für den Systemstart mit solch einer hybriden Speicherarchitektur sind in Abbildung 2.30 gezeigt.

Als Nebenbedingung für einen solchen Start muss der Systemdesigner, in Anlehnung zum Vorgang bei der Konfiguration, die Entscheidung treffen, welche Teile der Software zeitkritisch sind und schnellstmöglich starten müssen, und für welche Teile der Software eine Verzögerung in der Verfügbarkeit hingenommen werden kann. Die zeitliche Abfolge der verschiedenen Software Tasks ist beispielhaft in Abbildung 2.31 dargestellt. Nach dem Start eines potentiell vorhandenen

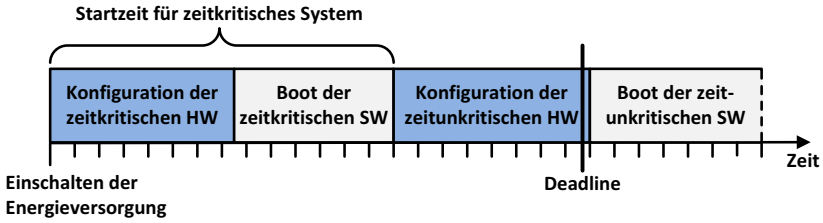


Abbildung 2.29.: Kompletter Systemstart mit sequentiellem Fast Startup Konzept für die Hardware Konfiguration als auch für das Booten der Software. Für die zeitkritischen Komponenten können Fristen wesentlich besser eingehalten werden als für den Start des Gesamtsystems.

Betriebssystems kann die zeitkritische Software direkt starten. Hierbei handelt es sich meist um Kommunikations- und Initialisierungsaufgaben.

Für einen Fast Startup Startvorgang werden außer den eigentlichen zeitkritischen Tasks der Anwendung zwei zusätzliche Tasks benötigt. Der erste kann sofort nach Start des Betriebssystems parallel mit dem Anwendungstask beginnen und ist für die zweite Phase der Hardwarekonfiguration zuständig. Er liest also mit Hilfe des entsprechenden Speichercontrollers den zweiten partiellen Bitstrom aus dem nichtflüchtigen Speicher und schreibt diesen mit Hilfe der internen Konfigurationsschnittstelle (Internal Configuration Access Port, ICAP) in den Konfigurationsspeicher des FPGAs, um die zeitunkritische Hardware zu implementieren. Daraufhin kopiert der zweite zusätzliche zeitkritische Task die zeitunkritischen Softwareteile in den Arbeitsspeicher, auf welche nun zugegriffen werden kann. Abschließend können nun auch die zeitunkritischen Tasks gestartet werden, das System ist vollständig verfügbar.

Eine Orientierungshilfe zur Trennung der Software ist, dass jegliche Software zur Ansteuerung für Hardware aus der zeitunkritischen Partition P-II natürlich selbst auch zeitunkritisch ist. Zeitkritisch sind außer den zwei zusätzlichen Routinen normalerweise, falls vorhanden, das Betriebssystem sowie viele der Routinen zur Ansteuerung der Hardware aus der zeitkritischen Partition P-I. Welche Routinen im Detail alle zeitkritisch sind hängt von der konkreten Anwendung ab. Weitere Informationen hierzu finden sich in [MNH+13].

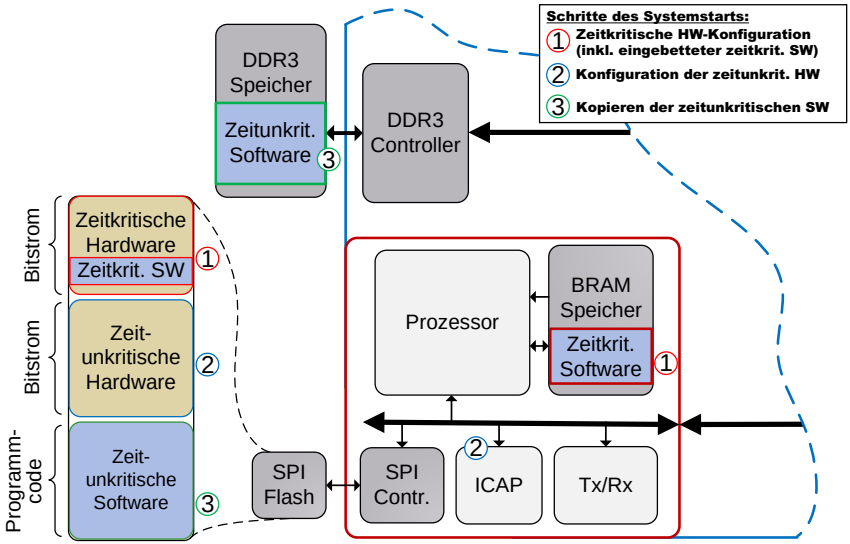


Abbildung 2.30.: Ablauf des Starts mit hybrider Speicherarchitektur: Zuerst werden in Schritt 1 die zeitkritische HW&SW durch die Konfigurationlogik des FPGAs geladen. In Schritt 2 rekonfiguriert der Prozessor den noch unkonfigurierten Bereich mit dem zweiten Bitstrom. Im dritten Schritt wird die zeitunkritische Software in den DDR3 Speicher kopiert.

2.2.8. On-Chip Logikanalysator

Wie bei Software entstehen auch beim Entwurf von Hardware Schaltungen Fehler, welche gefunden und beseitigt werden müssen. Bei einer digitalen Schaltung aus dedizierten Bauteilen können mit Hilfe eines Logikanalysators die Ein- und Ausgangssignale einer jeden Komponente aufgezeichnet und analysiert werden. Bei einem FPGA ist dies nicht so einfach, da es nicht möglich ist, mit externen Messgeräten die Signale innerhalb eines FPGAs abzugreifen.

Vor der Realisierung eines Hardware Designs in einem FPGA wird dieses während der Entwicklungsphase ausgiebig auf Register-Transfer-Ebene simuliert. Korrekt beschrieben, verhält sich das Design anschließend im FPGA so, wie in der Simulation. Allerdings kommt es vor, dass Fehler in der Beschreibung oder auch in der Simulation oder den Synthesewerkzeugen dazu führen, dass sich eine Schaltung im FPGA nicht verhält, wie erwartet. Des Weiteren existieren bestimmte

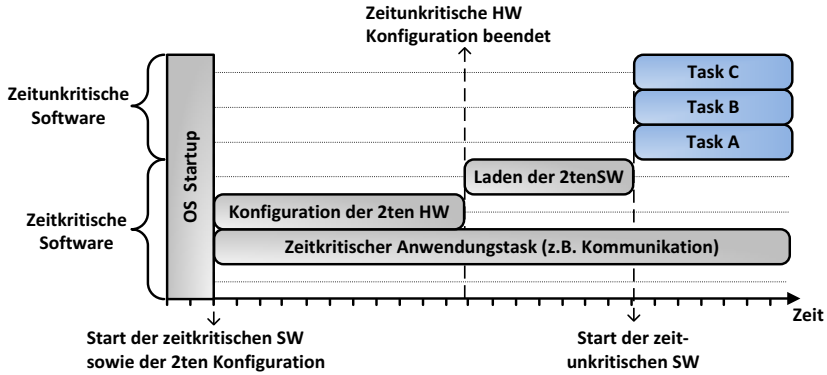


Abbildung 2.31.: Übersicht über die zeitliche Abfolge der verschiedenen Softwaretasks nach Start des Betriebssystems.

Strukturen, beispielsweise Schaltungen mit unabhängigen Taktbereichen, welche eine Simulation, die alle Fälle abdeckt, schwierig bis unmöglich macht.

Eine sehr große Hilfe um ein FPGA-Design von Fehlern zu befreien (debuggen) sind on-Chip, also auf dem Chip integrierte, Logikanalysatoren. Hierbei handelt es sich bei FPGAs allerdings nicht um fest verdrahtete, dedizierte Einheiten, sondern um einen *Intellectual Property Core* (IP-Core), welcher die Funktion eines Logikanalysators realisiert und der zu debuggenden Schaltung hinzugefügt wird. Mit ihm ist es möglich, anhand von verschiedenen Triggerbedingungen beliebige Signale innerhalb eines FPGAs zur Laufzeit aufzuzeichnen und diese dann über eine Schnittstelle, meist JTAG, komfortabel auszulesen. Alle größeren FPGA Hersteller stellen solche IP-Cores sowie die zugehörige Software zur grafischen Anzeige zur Verfügung.

Die Logikanalysator IP-Cores sind sehr umfangreich konfigurierbar. So können in der Regel die Anzahl der Signale, welche aufgezeichnet werden sollen, sowie die Anzahl der Samples im Prinzip frei gewählt werden. Des Weiteren kann die Triggerlogik durch zusätzliche FPGA Logik stark erweitert werden. Einzige Grenze sind die Ressourcen, die der Logikanalysator zur Realisierung benötigt. So sind je nach Anzahl an Signalen und Speichertiefe eine Vielzahl an BRAM-Primitive notwendig.

Ebenso sollte bedacht werden, dass es einen hohen Aufwand erfordert, um zu verhindern, dass das Hinzufügen des Logikanalysator IP-Cores Einfluss auf die Implementierung der Schaltung nimmt. So kann es trotz der Hilfe der Analyse-

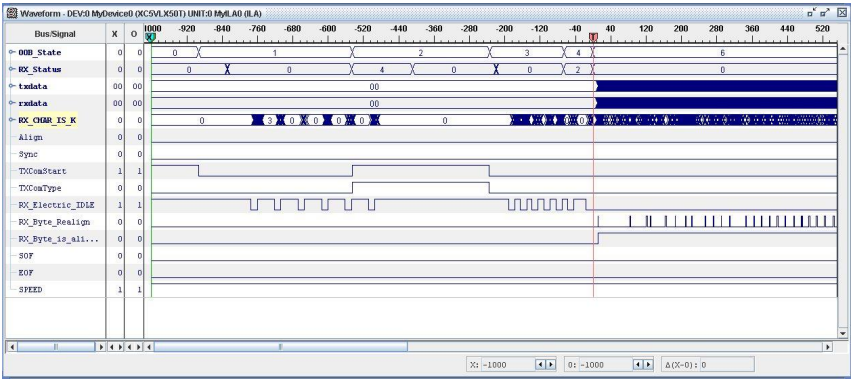


Abbildung 2.32.: Grafische Benutzeroberfläche von Chipscope, dem on-Chip Logikanalysator der Firma Xilinx [98].

tools durch die Herstellerwerkzeuge sein, dass das Hinzufügen die Laufzeit bestimmter Signal günstig oder ungünstig beeinflusst.

Die Software und IP-Core Sammlung, welche einen on-Chip Logikanalysator beinhaltet und welche die Firma Xilinx für ihre FPGAs zur Verfügung stellt, nennt sich *Chipscope*. Neben dem Logikanalysator bietet Chipscope viele weitere Möglichkeiten, um beispielsweise Signale auf dem Chip zu generieren oder um die High-Speed Transceiver der FPGAs zu testen. Weitere Informationen zu Chipscope finden sich im entsprechenden Userguide [104] und Datenblatt [99]. Abbildung 2.32 zeigt die typische Ansicht der Chipscope Software zur Darstellung der Signale.

2.3. Architekturen zur digitalen Signalverarbeitung für OFDM

Dieses Kapitel soll die verschiedenen Möglichkeiten integrierter Schaltungen auf Architekturebene aufzeigen, die zur optimierten OFDM-Signalverarbeitung zur Verfügung stehen. Das Ziel ist, die Vor- und Nachteile der verschiedenen Architekturen in Bezug auf die Verwendung zur digitalen Signalverarbeitung herauszustellen. Dabei soll eine möglichst hohe Leistung bei hoher Flexibilität erreicht werden. Andere Kriterien wie der Energieverbrauch oder die Kosten spielen eine untergeordnete Rolle. Die hier zusammengefassten sowie weiterführenden Informationen finden sich, falls nicht anders vermerkt, in [45], [14], [50] und [73].

2.3.1. General Purpose Processors (GPP)

Standard Mikroprozessoren, auch *General Purpose Processors* (GPP) oder *Central Processing Unit* (CPU) genannt, sind nicht auf ein spezielles Anforderungsgebiet optimiert und daher am weitesten verbreitet. Sie werden seit Jahren weiterentwickelt und verbessert. Aus diesem Grund sind für diese Art von Prozessoren zum einen die Entwicklungswerkzeuge am weitesten fortgeschritten, zum anderen kann zur Programmierung auf moderne Hochsprachen und umfangreiche Bibliotheken zurückgegriffen werden. Es existieren sogar mächtige grafische Programmiersysteme wie beispielsweise *LabVIEW*, mit welchen sich Programmcode anhand von Datenflussmodellen generieren lässt.

Die Möglichkeiten des Programmentwurfs solcher Prozessoren ermöglichen die einfache Erstellung und Inbetriebnahme komplexer Systeme, ohne dass hierfür notwendigerweise tiefgehende Kenntnisse des Hardware- und Softwareentwurfs benötigt werden. Mit ihnen können theoretisch alle Arten von OFDM-Systemen entworfen werden, solange die zu realisierende Datenrate nicht die Leistungsfähigkeit des Prozessors übersteigt. Parameteränderungen benötigen für ein GPP-basiertes System lediglich das Ändern von Software und lassen sich somit sehr einfach und schnell bewerkstelligen.

Aufgrund dieser Vorteile ist ein Trend zur Verwendung von GPPs im Bereich der Signalverarbeitung zu beobachten, vergleiche [119]. Der Nachteil der limitierten maximalen Datenrate gegenüber auf DSP optimierte Architekturen, gerade bei komplexen Signalverarbeitungsaufgaben, erfordert im Hochgeschwindigkeits-Bereich allerdings den Einsatz von Alternativen, vergleiche [27] und [16].

2.3.2. Grafikprozessoren

Grafikprozessoren (Graphics Processing Unit, GPU) sind die wohl bekanntesten Spezialprozessoren. Angefangen mit dem Aufkommen grafischer Betriebssysteme wurden sie entwickelt, um den Hauptprozessor zu entlasten. Nicht zuletzt durch den Markt für Computerspiele werden GPUs schneller weiter entwickelt, als andere Spezialprozessoren.

Im Gegensatz zu einer modernen CPU, welche aus einigen wenigen aber hochkomplexen Prozessorkernen besteht, finden sich in einer GPU hunderte, sehr einfache Rechenkerne, siehe Abbildung 2.33. Dementsprechend eignen sich Grafikprozessoren, je nach Komplexität des Algorithmus, sehr gut für hochparallelisierbare Aufgaben. Beispiele hierfür sind Algorithmen der Bildverarbeitung als auch der Signalverarbeitung.

Die Verwendung von GPUs in Bereichen außerhalb der Bilddarstellung wurde in den letzten Jahren zunehmend gefördert, beispielsweise durch die Entwicklung

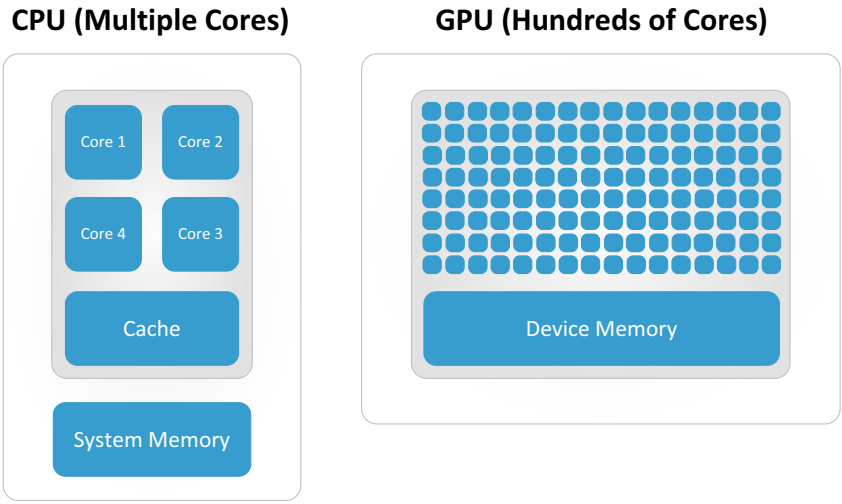


Abbildung 2.33.: Vergleich der Anzahl von Prozessorkernen zwischen einer CPU und einer GPU [76].

spezieller Programmierschnittstellen. So gibt es GPUs, welche für den Einsatz als Mehrzweck-Coprozessor optimiert wurden, beispielsweise durch die Verwendung typischer Wortgrößen. Bei solch einem Einsatz einer GPU spricht man von *General Purpose Computation on Graphics Processing Unit* (GPGPU).

Trotz allem existieren auch Nachteile. So müssen für genügend Performanz möglichst viele Rechenkerne auch genutzt werden können, die Anwendung muss also entsprechend parallelisierbar sein. Des Weiteren stellt die Anbindung an die CPU, typischerweise über den *Peripheral Component Interconnect Express Bus* (PCI-Express), meist einen Flaschenhals für das GPGPU dar. Weitere Informationen zur Architektur von GPUs finden sich beispielsweise im Anhang von [73].

2.3.3. Digitale Signalprozessoren

Während *digitale Signalprozessoren* (Digital Signal Processor, DSP) oftmals allgemein als digitale Signale verarbeitende Mikroprozessoren definiert werden, sind damit meist Prozessoren gemeint, welche speziell auf Anforderungen der Signalverarbeitung hin entwickelt wurden. Wie bei den GPUs handelt es sich um Spezialprozessoren, allerdings nicht zur Grafikdarstellung, sondern für die Signalver-

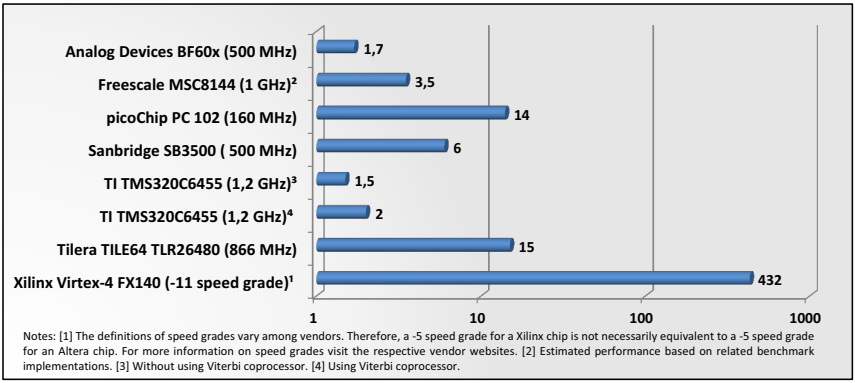


Abbildung 2.34.: BDTI OFDM Empfänger Benchmark Ergebnisse für Konfigurationen hoher Kapazität (maximale Anzahl der unterstützten BDTI Kanäle des Systems). Logarithmische Skalierung beachten. ©Berkeley Design Technology, Inc. (BDTI) Used with permission [13].

arbeitung optimiert. Fast alle diese Prozessoren basieren auf einfachen Harvard-Architekturen, welche darauf ausgelegt sind, sich wiederholende arithmetische Operationen effizient und eventuell auch mehrfach parallel auszuführen.

DSPs besitzen meist mehrere dedizierte Multiplikationsakkumulatoren, da in Algorithmen der Signalverarbeitung die Multiplikationsakkumulator-Operation besonders häufig benötigt wird (z.B. für digitale Filter). Ebenso werden schnelle Schnittstellen zu einem externen Speicher (realisiert durch mehreren Speicherbänke) sowie Schnittstellen zu externen Komponenten wie Digital/Analog-Wandler oder Analog/Digital-Wandler realisiert. Sehr oft verwenden DSPs auch ein Festkomma-Format zur Darstellung von Zahlen, wodurch viele DSPs unübliche Wortbreiten besitzen. Details zu DSP Architekturen finden sich in [14].

Während DSPs Signalverarbeitungsaufgaben schneller und effektiver durchführen als Standard Mikroprozessoren, liegen sie dennoch in Bezug auf die Leistungsfähigkeit deutlich hinter dedizierten Schaltungen zurück. In Abbildung 2.34 werden die Ergebnisse eines OFDM Empfänger Benchmarks von verschiedenen DSPs mit dem Ergebnis einer FPGA Implementierung verglichen, siehe [13].

2.3.4. Prozessor mit anwendungsspezifischem Befehlssatz

Die Überlegenheit innerhalb von Anwendungen der Signalverarbeitung gegenüber GPPs erzielen die DSPs durch die Optimierung für diesen Bereich. Allerdings bringen diese Optimierungen Einschränkungen in der Flexibilität mit sich. Aufgaben außerhalb der Signalverarbeitung sind nur schwer realisierbar. Für eine noch stärkere Optimierung auf ein beliebiges aber bestimmtes Anwendungsgebiet, können sogenannte Prozessoren mit anwendungsspezifischem Befehlssatz (Application-specific instruction-set processors, ASIPs) entwickelt werden, wobei natürlich auch DSPs zu den ASIPs gezählt werden können.

Wie der Name schon nahe legt verfügt ein ASIP über einen sehr angepassten Befehlssatz, der gerade genügend Flexibilität bietet, um für das anvisierte Anwendungsgebiet genügend Variationen abzudecken. Das Ziel eines ASIPs ist es, die größtmögliche Leistung in Bezug auf Kosten und Energieverbrauch zu erreichen. Der Flexibilität werden nur soweit wie nötig Zugeständnisse gemacht. Dabei werden für ASIPs entweder einfache existierende Architekturen angepasst oder von Grund auf neue Mikroarchitekturen entworfen, um den ASIP optimal auf die Anwendung beziehungsweise den Algorithmus zuzuschneiden, siehe beispielsweise [68].

Der Kompromiss zwischen Flexibilität und Leistungsfähigkeit ist dabei von Fall zu Fall unterschiedlich, eine grobe Eingliederung ist dennoch in Abbildung 2.35 zu sehen. Das Bild zeigt auch, dass es noch andere Faktoren wie Performanz und Flexibilität gibt, diese jedoch Abhängigkeiten zueinander besitzen. So steigen die Kosten und die Produkteinführungszeit (Time-To-Market) mit der Leistungsfähigkeit, der Energieverbrauch hingegen steigt mit der Flexibilität.

2.3.5. Coprozessoren und Hardware-Beschleuniger

Coprozessoren stellen programmierbare Hardware-Beschleuniger dar, deren prinzipieller Aufbau dem von ASIPs ähnelt, vgl. [45]. Auch sie verwenden eine instruktionsbasierte Zuweisung von Operanden des internen Datenpfads zur Signalverarbeitung, allerdings ist ihr Anwendungsgebiet im Vergleich zu ASIPs wesentlich eingeschränkter.

Eine genau Abgrenzung zwischen Coprozessor und Hardware-Beschleuniger existiert nicht, der Begriff wird oft synonym verwendet. So werden auch häufig Grafikprozessoren als eine Art Coprozessor bzw. Hardware-Beschleuniger gesehen. Essentiell für einen solchen Beschleuniger ist, dass er einem Hauptprozessor untergeordnet ist, welcher die Ansteuerung des Beschleunigers übernimmt.

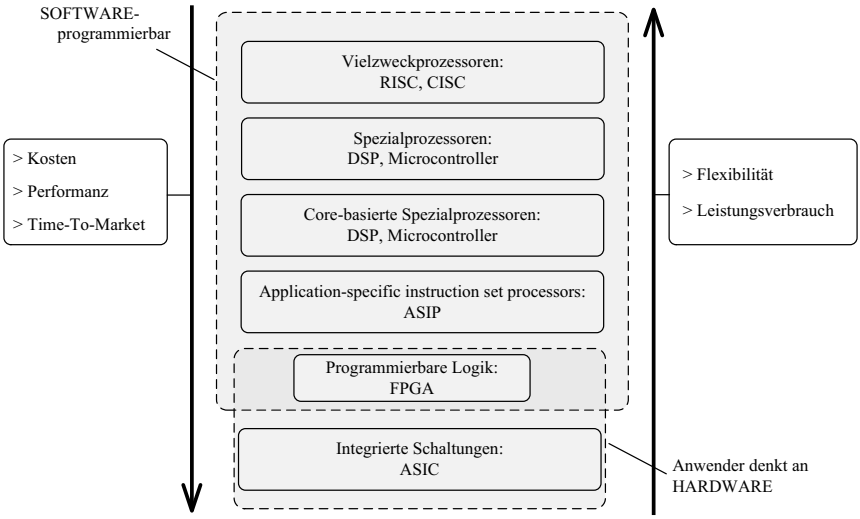


Abbildung 2.35.: Durch den Verzicht auf Flexibilität lassen sich mit entsprechenden Optimierungen für ein Anwendungsgebiet enorme Leistungssteigerungen erzielen. Das Extrembeispiel ist der dedizierte Schaltkreis für das entsprechende Problem, realisiert in einem ASIC. [36]

2.3.6. Multiprocessor System-on-Chip (MPSoC)

Wie bei den Mikroprozessoren kommen auch die DSPs und ASIPs langsam an die Grenzen der erzielbaren Leistung einzelner Prozessoren. Da sich in der Signalverarbeitung jedoch sehr viele Algorithmen sehr gut parallelisieren lassen, sind Multiprozessorsysteme in diesem Bereich besonders effizient und bieten somit auch hier einen Ausweg. Ein System, welches sich gänzlich oder zu großen Teilen auf einem einzigen Chip befindet und dabei über mehrere Prozessoren verfügt, wird Multicore System-on-Chip beziehungsweise Multiprozessor System-on-Chip (MPSoC) genannt.

Bei der Klassifizierung als MPSoC spielt es keine Rolle, ob es sich bei den Prozessoren um GPPs, DSPs oder ASIPs handelt. Oft werden auch verschiedene Architekturen verwendet, um in solch einer heterogenen Struktur dann die Vorteile der einzelnen Konzepte zu vereinen. So findet sich beispielsweise in einem modernen Smartphone System-on-Chip wie dem Snapdragon 800 oder Apples A7 fast immer ein Arm Prozessoren mit mehreren Kernen für allgemeine Aufgaben,

ein DSP zur Signalverarbeitung sowie ein Grafikprozessor (Graphics Processing Unit, GPU) für die Grafikdarstellung. Für weitere Ausführungen des Themas MP-SoC für DSP Berechnungen sei auf [14] verwiesen.

2.3.7. ASICs und FPGAs

Bei einem ASIC handelt es sich um eine *anwendungsspezifische integrierte Schaltung* (Application-specific integrated circuit, ASIC), also eine Schaltung, deren Funktion im Gegensatz zu universellen integrierten Schaltungen wie CPUs, nicht mehr vom Anwender verändert werden kann. Während alle bisher genannten Architekturen durch Software programmierbar sind und sich entsprechend in Abbildung 2.35 im oberen Kasten befinden, wird die höchste Performanz durch solche dedizierte Schaltungen erreicht.

Bei ASICs handelt es sich um festverdrahtete Schaltkreise, welche die direkte Umsetzung von Anwendungen oder Algorithmen in Hardware darstellen, siehe [45]. Sie sind nicht programmierbar, sondern genau für einen Anwendungszweck entworfen. Dadurch kann die Schaltung vollständig auf diese Anwendung optimiert werden. Flexibilität besitzt ein ASIC allerdings in der Regel nicht. Falls ein ASIC doch die Möglichkeit einer Konfiguration bietet, sind meist mehrere Varianten parallel realisiert und der Anwender kann durch das Steuern von Multiplexern bestimmen, welche Variante verwendet werden soll.

FPGAs sind in erster Linie rekonfigurierbare Bausteine, welche digitale Schaltungen realisieren können. Mit ihnen kann die Umsetzung einer Schaltung auch stattfinden, falls sich die Fertigung als ASIC aus Kosten- und/oder Zeitgründen nicht lohnt. Allerdings lassen sich mittels eines FPGAs auch programmierbare Schaltungen, beispielsweise Prozessoren, implementieren, weshalb man sie zu beiden Gruppierungen, Hardware als auch Software, zählen kann. Sie ermöglichen, aus den selben Gründen wie die ASICs, eine enorme Performanz, allerdings besitzen sie durch ihre Architektur und ihre Rekonfigurierbarkeit immer einen Overhead gegenüber ASICs.

3. Stand der Technik

Im folgenden Kapitel soll der Stand der Technik zusammengefasst werden, welcher für die vorliegende Arbeit und deren Ziele relevant ist. Zunächst folgt ein Überblick über OFDM-Signalverarbeitungssysteme und ihre Eigenschaften, um anschließend bestehende Arbeiten unterschiedlicher Varianten von OFDM-Systemen auf diese Kriterien hin zu untersuchen. Während dabei Sender- und Empfängerimplementierung im Fokus liegen schließt Kapitel 3.3 den Stand der Technik, mit einem speziellen Beitrag zur Digital Down Converter Realisierungen, welche oftmals für die Vorverarbeitung von OFDM Signalen zum Einsatz kommen, ab.

3.1. OFDM Signalverarbeitung in der Kommunikationstechnik

Durch den stetigen Fortschritt der digitalen Signalverarbeitung werden komplexe Modulationsverfahren wie OFDM, welche sich in der drahtlosen Übertragung bereits fest etabliert haben, auch für Hochgeschwindigkeits-Kommunikationssysteme interessant. Viele Forschergruppen beschäftigen sich daher mit den Vor- und Nachteilen solcher Techniken für optische Datenübertragung, wie sie beispielsweise in aktuellen und zukünftigen Zugangsnetzwerken zu finden ist. Doch trotz einer enormen Leistungssteigerung in diesem Bereich ist die maximal erreichbare Datenrate eines solchen Systems in der Regel immer noch durch die erreichbare Bandbreite der Elektronik und durch die Limitierungen der Signalverarbeitung beschränkt.

Während Experimente bereits belegen, dass durch optische Systeme Datenraten von mehreren Terabit pro Sekunde möglich sind [HSS+11], bewegen sich die erreichten Datenraten für Hochgeschwindigkeits-Systeme mit Echtzeit-Signalverarbeitung sowohl für Sender als auch für Empfänger im Gigabit-Bereich. Infolgedessen besteht ein Schwerpunkt der Forschung darin, die erreichbare Datenrate für entsprechende Systeme zu maximieren. Diese Maximierung des Durchsatzes wird vor allem in der optischen Kommunikationstechnik angestrebt. Zu diesem Zweck wird dort in der Regel die Echtzeitverarbeitung der Signale mit dedizierten Schaltungen durchführt, welche mit Hilfe von FPGAs realisiert werden, vergleiche Kapitel 3.2.2.

In allgemeinen Anwendungsfällen werden OFDM-Systeme meist nicht nur für eine maximale Datenrate optimiert, da häufig auch andere Eigenschaften eines solchen Signalverarbeitungssystems eine entscheidende Rolle spielen. So spielt beispielsweise gerade für mobile Geräte wie Smartphones eine Reduzierung des Energieverbrauchs eine entscheidende Rolle.

Ein weiteres Kriterium ist die Flexibilität eines Systems. Zum einen kann bereits von einer einzelnen Anwendung gefordert werden, mehrere Parametereinstellungen wie verschiedene Modulationsarten und verschiedene Größen des Cyclic Prefix zu unterstützen, zum anderen bedeutet eine hohe Flexibilität ein größeres potentielles Einsatzgebiet. Dies führt zu einer höheren Stückzahl und so zu geringeren Kosten. Gerade als Mess- und Analysesystem ist eine Vielfalt an Parameter-Konfigurationsmöglichkeiten unerlässlich, um in einem Labor in der Lage zu sein, die Einflüsse verschiedenster Parameterkombinationen auf Konzepte und Bauteile zu untersuchen.

In allen Fällen ist es hilfreich, wenn das Einstellen bzw. Programmieren eines Systems einfach und ohne großes Spezialwissen möglich ist, damit solch ein System schnell aufgebaut ist und es auch interdisziplinär zur Anwendung kommen kann. Im Folgenden werden verschiedene Architekturen untersucht, welche sich im Kompromiss zwischen Leistungsfähigkeit und Flexibilität sowie in anderen Eigenschaften teilweise stark unterscheiden.

3.2. Architekturen zur OFDM-Signalverarbeitung

Neben dedizierten Schaltungen werden sowohl herkömmliche als auch spezialisierte Rechenarchitekturen zur OFDM-Verarbeitung verwendet und evaluiert. Zur effizienten Verarbeitung digitaler OFDM-Signale sind Standard Mikroprozessoren auf Grund ihrer Performanz gerade im Hochgeschwindigkeitsbereich ungeeignet, vergleiche Abbildung 2.35. Wie bei digitalen Kommunikationssystemen oft üblich, muss ein konstanter Strom an Daten durchgehend verarbeitet werden. Während diesen Verarbeitungsschritten muss in der Regel auf wenige Ereignisse reagiert werden. Der Ablauf der einzelnen Schritte ist fest vorgegeben.

Da, außer wenige Ausnahmen, die einzelnen Daten keine Abhängigkeiten aufweisen, lassen sich die meisten Algorithmen und Berechnungen für OFDM gut parallelisieren. Entsprechend ist die OFDM Signalverarbeitung auch gut für DSP-Architekturen geeignet, welche durch Optimierungen wie Single-Instruction-Multiple-Data (SIMD) Befehle, die Parallelisierung von Algorithmen und Berechnungen unterstützen. Neben den bekannten DSP optimierten Architekturen wie zum Beispiel Grafikprozessoren (GPU) und Digitale Signalprozessoren (DSP) existieren allerdings bisher nur wenige anwendungsspezifische Architekturen, welche speziell auf OFDM zugeschnitten sind.

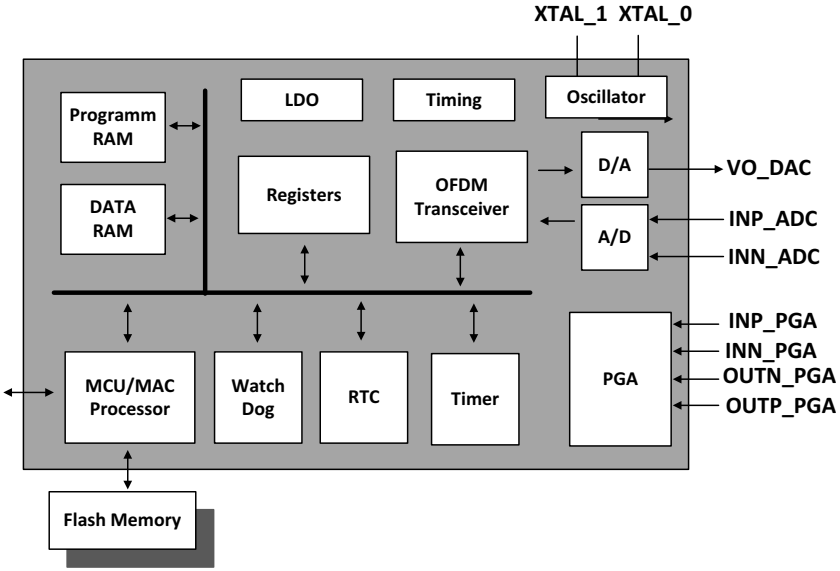


Abbildung 3.1.: Blockschaltbild des LME2980, ein industrielles SoC, welches OFDM für den Einsatz in Powerline Communication Systemen verwendet [1].

3.2.1. Industrielle OFDM-Architekturen

Neben den publizierten Realisierungen aus Forschung und Entwicklung ist wichtig, welche Arten von industriellen Chips es bereits fertig zu kaufen gibt. Doch ist es gerade für die neuesten Bausteine schwer, genauere Informationen zu erhalten, wodurch Rückschlüsse auf die Architektur nicht immer möglich sind.

Alle industriellen Chips, welche speziell auf OFDM ausgelegt sind, legen sich auf einen oder mehrere Industriestandards fest. So existiert beispielsweise von Leaguer MicroElectronics der *LME2980*, ein integriertes SoC zur Powerline Communication (PLC). Der Chip enthält einen kompletten OFDM-Transceiver, eine Fehlerkorrektur, ein analoges Front-End sowie einen Mikrocontroller mit 48 KB Programmspeicher. Er nutzt OFDM mit bis zu 1280 Subträgern, die Subträger können mit BPSK, QPSK und 16-QAM moduliert werden. Der Chip erreicht dadurch Datenraten bis zu 306 Kb/s. Das Blockschaltbild des LME2980 ist in Abbildung 3.1 dargestellt, ein ähnliches SoC, mit etwas niedrigerer Datenrate von max. 128 Kb/s ist der ST7590 von ST Microelectronics.

Im WLAN-Bereich ermöglicht der Chip AR9462 die Realisierung des aktuellen IEEE 802.11n Standards mit bis zu 2 Datenströmen. Er erreicht durch die OFDM-Technologie mit 114 Subträgern (108 Datenträger), welche per BPSK, QPSK, 16-QAM oder 64-QAM moduliert sind, eine Datenrate von bis zu 300 Mb/s. Die Schnittstelle zum Host ist durch PCI-Express gegeben. Der Nachfolgerchipsatz QCA9880 von Qualcomm Atheros, welcher für die Realisierung des kommenden IEEE 802.11a/c Standards vorgesehen ist, wird bis zu 242 Subträger (234 Datenträger) unterstützen. Diese sollen mittels BPSK, QPSK, 16-QAM, 64-QAM und 256-QAM moduliert werden können, was in einer 3x3 MIMO Konfiguration bis zu 1,3 Gb/s erlauben wird.

Keiner der zu Beginn dieser Arbeit erhältlichen Chips erfüllte die gewünschten Datenraten im Gigabit Bereich; auch die neuesten Versionen erreichen diese nur mit Mühe. Da außerdem keine näheren Informationen zu der Signalverarbeitungsarchitektur der vorgestellten Chips erhältlich ist, ist nur schwer zu beurteilen, wie stark ihre Architektur auf die entsprechenden Standards zugeschnitten ist. Es liegt jedoch nahe, dass es nicht möglich ist, mit ihnen Systeme außerhalb dieser Standards zu realisieren, die Flexibilität daher nicht den gewünschten Anforderungen entspricht.

3.2.2. Dedizierte Schaltungen und FPGAs

Die durchsatzstärksten Sender und Empfänger, welche OFDM Signale in Echtzeit verarbeiten, finden sich in den Experimenten mit optischer Signalübertragung und nutzen mittels FPGA realisierte, dedizierte Schaltungen. Da ein OFDM-Sender im direkten Vergleich mit einem Empfänger das einfachere System darstellt, finden sich deutlich mehr Publikationen über Echtzeit OFDM-Sender, welche im Allgemeinen auch höhere Datenraten erzielen.

Während 2011 eine Forschungsgruppe bei Nokia Siemens Networks einen Sender mit einer Datenrate von 46,9 Gb/s (93,8 Gb/s mit 2 Polarisationen) zeigte [40], gelang unserer Forschungsgruppe im selben Jahr die Demonstration einer dedizierten OFDM Sender Implementierung, welche eine Datenrate von 101,5 Gb/s ermöglichte [SWN+11]. 2 Jahre später wurde diese Rekordrate durch den in [81] vorgestellten Nyquist Pulse Sender, einer OFDM verwandten Modulationsformat, auf 126 Gb/s erhöht, wobei dieser die gleiche Architektur, also mittels FPGA realisierte, dedizierte Schaltungen verwendet.

Der zusätzliche Aufwand, welcher durch die Synchronisation und die Kanalkorrektur entsteht, erschwert die Realisierung von OFDM-Empfängern. 2010 demonstrierte Alcatel Lucent einen Echtzeit OFDM Empfänger mit 3,55 Gb/s [47]. Dabei kamen eigens für OFDM entwickelte, dedizierte Schaltungen zum Einsatz,

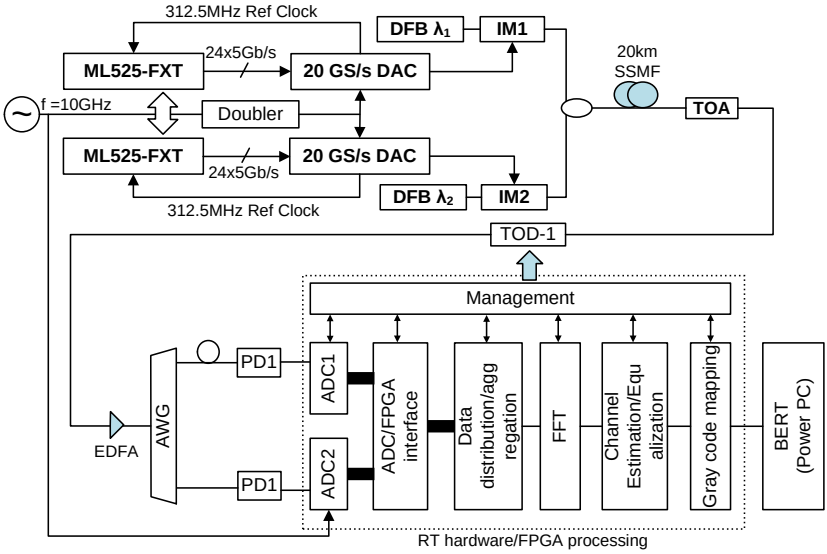


Abbildung 3.2.: 41,25 Gb/s Rekord-Empfänger Experiment von 2010 der Firma NEC. Der Empfänger mit Echtzeitsignalverarbeitung realisiert einen dedizierten Aufbau in einem FPGA, er verwendet eine FFT der Größe 32. Es wird die Taktquelle des offline-Sender verwendet [75].

die mittels FPGAs realisiert wurden und neben der Modulationsart der Subträger keine Möglichkeiten bieten, Parameteränderungen vorzunehmen.

Das Blockschaltbild eines 41,25 Gb/s Experiments der Firma NEC, auch aus dem Jahre 2010 ist in Abbildung 3.2 dargestellt. Als Sender kommt ein Arbiträrgenerator zum Einsatz, welcher mit Hilfe von 2 Virtex-5 FPGA Prototyping Boards aufgebaut wurde (ML525-FXT). Der Empfänger verwendete die selbe Taktquelle wie der Sender, was Vorteile bietet, da es beispielsweise zu keinem Versatz der Samplingfrequenzen kommen kann. Dies erlaubt es, auf eine Korrektur der Samplingfrequenz zu verzichten und trotzdem die zeitliche Synchronisation nur einmal zur Initialisierung des Systems durchzuführen, was die Synchronisierung erheblich vereinfacht. Synchronisiert wurde mit einem Tunable Optical Delay (TOD). Diese Vereinfachungen durch die Verwendung der selben Taktquelle sind bei praktischen OFDM-Systemen außerhalb des Labors in der Regel nicht möglich.

Der Empfänger verarbeitet 192 komplexe Samples pro Takt. Die Algorithmen sind daher hochparallel als dedizierte Schaltungen realisiert, beispielsweise kommen sechs FFT-Blöcke der Firma Xilinx zum Einsatz. Die Demodulation der Subträger unterstützt QPSK, 8-QAM sowie 16-QAM. Weitere Details zum Aufbau und zu den Ergebnissen finden sich in [75].

3.2.3. Grafikprozessoren und Digitale Signalprozessoren

Grafikprozessoren werden längst nicht nur zur Berechnung von Bildschirmausgaben eingesetzt sondern sie haben sich im Laufe der Zeit zu einem echten Coprozessor für rechenintensiven, parallelisierbaren und durch den Datenfluss dominierten Aufgaben entwickelt. So existieren eigene Programmierschnittstellen (z.B. CUDA der Firma Nvidia, oder die Open Computing Language), welche die einfache Entwicklung von Programmen für diese Architekturen unterstützen. In [69] sowie in [88] wurden MIMO-OFDM Systeme, also OFDM-Systeme mit mehreren Kanälen, mit Hilfe von Grafikprozessoren aufgebaut. Dabei wurde im Vergleich zu einer CPU Realisierung eine Erhöhung des Durchsatzes um den Faktor 10 erreicht, was zu Datenraten bis zu 100 Mb/s führte. Parameteränderungen sind hierbei natürlich leicht durch Ändern des Programmcodes zu erzielen. Im Jahr 2013 veröffentlichten Ma et al. in [57] ein System, welche mit der Hilfe von drei GPUs eine Datenrate von 476 Mb/s erreichte.

Auch DSPs eignen sich gut um OFDM-Signalverarbeitung zu betreiben. Ähnlich wie bei den Grafikprozessoren liegt der große Vorteil solcher Architekturen darin, dass sie enorm flexibel programmierbar sind und praktisch alle relevanten Parameter leicht durch Software geändert und angepasst werden können. In [116] wird die Realisierung eines MIMO-OFDM Unterwasser Akustik-Kommunikationssystems einmal mit einem DSP mit Floating Point Zahlendarstellung und einmal mittels Festkomma DSP beschrieben. Ebenso wie in [117] wird der IEEE 802.11n Standard implementiert, in beiden Veröffentlichungen kommen DSPs der Firma Texas Instruments zum Einsatz, die erreichten Datenraten liegen bei maximal 600 Mb/s. Wie auch schon in Abbildung 2.34 des Grundlagenkapitels gezeigt, liegen sie in Bezug auf die maximale Datenrate damit deutlich hinter den dedizierten Schaltungen zurück.

3.2.4. HW-Beschleuniger, ASIPs und MPSoCs für OFDM

Gerade für die Fast Fourier Transformation wurden schon sehr früh Prozessoren und Beschleuniger entwickelt, da dieser Algorithmus in sehr vielen Anwendungen benötigt wird, sich gut parallelisieren und sich effizient in Hardware umsetzen lässt. Bereits Ende der 60er Jahre gab es erste Veröffentlichungen zu die-

sem Thema [12]. Das Interesse an FFT-Prozessor Realisierungen, welche mit Hilfe der neuesten Technologien und Implementierungstechniken stetig leistungsfähiger werden, lässt nicht nach, vgl. [58].

Insgesamt lassen sich FFT-Beschleuniger in zwei Kategorien einteilen. So gibt es Coprozessoren, die ihren Datenpfad stark auf den FFT-Algorithmus auslegen, beispielsweise [43] und [32]. Durch diese Optimierungen erreichen sie in der Regel eine höhere Performanz oder Effizienz. Andere Prozessoren wie [92] versuchen den Datenpfad allgemeiner zu halten und die Abarbeitung des Algorithmus lediglich durch eine Parallelisierung der Berechnungen zu beschleunigen. Dies hat den Vorteil, dass sich solche Prozessoren auch für andere Algorithmen mit ähnlichen Anforderungen verwenden lassen.

Zusätzlich bieten alle großen FPGA-Hersteller bereits fertige FFT-Umsetzungen in ihren IP-Core Bibliotheken an. Diese lassen sich entweder direkt als Beschleuniger an die vom Hersteller favorisierte Busschnittstelle anbinden oder man nimmt sie als Basis für eigene Entwicklungen, vergleiche [114] und [37].

Abgesehen von FFT-Beschleunigern sind Veröffentlichungen über OFDM spezifische Beschleuniger und Coprozessoren rar. Dennoch, in [33] wird ein Prozessor vorgestellt, welcher neben der Berechnung der FFT ebenso eine Kanalschätzung beschleunigen kann. Ein anderer Prozessor zur Kanalschätzung wurde 2010 vorgestellt [8]. Er unterstützt durch seinen rekonfigurierbaren Datenpfad nicht nur verschiedene Schätzverfahren, er kann durch Erweiterungen auch zur Synchronisation eingesetzt werden.

Ein Beschleuniger welcher speziell für die Synchronisation von OFDM-Systemen entwickelt ist, wurde 2011 in [2] veröffentlicht. Allerdings kümmert sich dieser nicht um die Detektion von Paketen, eine der Hauptaufgaben der Synchronisation. Dies soll durch zusätzliche Logik vorangehend geschehen.

2006 wurde in einer Dissertation eine komplette Empfängerarchitektur veröffentlicht, siehe [35]. Es handelt sich um MPSoC welches auf COFFEE basiert, einem quelloffenen RISC Prozessor der Universität Tampere [52]. Dieser bietet einen dedizierten Bus für bis zu vier Hardwarebeschleuniger. An diesen Bus sind ein I/O-Coprozessor, ein Synchronisations-Coprozessor und ein FFT-Coprozessor (wobei in der Arbeit als Demodulation-Coprozessor bezeichnet) angeschlossen, die Systemarchitektur wird *Espresso* genannt und ist in Abbildung 3.3 dargestellt.

Durch eine Gruppe der Firma Philips wurde in [31] ein MPSoC entwickelt, welches aus mehreren, für bestimmte Algorithmengruppen spezialisierten Prozessoren besteht. Hierdurch konnten verschiedene Standards wie T-DMB und IEEE 802.11a realisiert werden, was Datenraten im Bereich zweistelliger Megabit pro Sekunde entspricht.

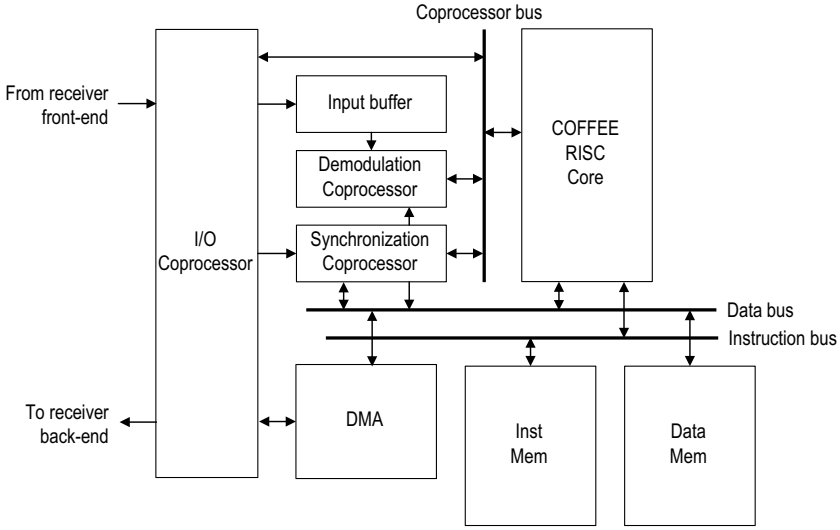


Abbildung 3.3.: Die Espresso Plattform basiert auf einem quelloffenen RISC-Prozessor namens *COFFEE*, an welchen über einen dedizierten Bus drei Coprozessoren angebunden sind.

In [91] wurde hingegen ein System-on-Chip entwickelt, welches eine dedizierte Implementierung des Standards IEEE802.11a als Ziel verfolgt und daher auch maximal Datenraten von 54 Mb/s erreicht. Dabei wurde der Fokus auf eine geringe Chipfläche und einen geringen Energieverbrauch gelegt.

Alle Systeme versuchen hierbei nicht direkt den größtmöglichen Datendurchsatz zu erzielen. Meist wird versucht nur eine Datenrate für einen bestimmten Standard zu erreichen, die Optimierung erfolgt anschließend zu Gunsten des Energieverbrauchs und der Chipfläche. Daher erreicht auch keines dieser Systeme Datenraten im Gigabit-Bereich.

3.3. Hochgeschwindigkeits Digital Down Converter (DDC)

Digital Down Converter (DDC) können eine wichtige Rolle innerhalb eines OFDM Systems spielen, vergleiche [18] und [89]. So kann beispielsweise durch ein Digital Down Converter eine Trägeruntergruppe aus einem OFDM Signal extrahiert

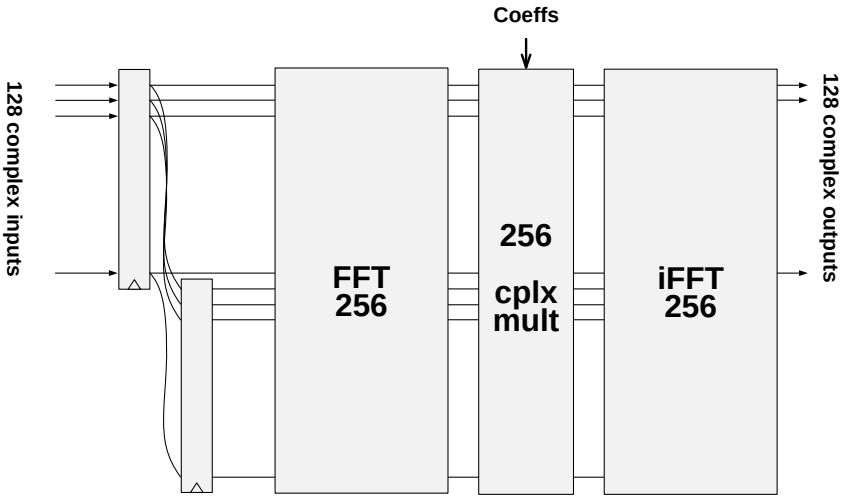


Abbildung 3.4.: 128-tab FIR-Filter, aufgebaut mit Hilfe der schnellen Faltung. Hierbei wird eine Multiplikation mit der Impulsantwort im Spektralbereich realisiert [24].

werden, um diese Untergruppe dann mit entspannten Anforderungen an die Signalverarbeitung weiter zu verarbeiten.

Hochgeschwindigkeits-DDC Realisierungen mit einer Bandbreite im zweistelligen Gigahertz-Bereich sind bisher nicht existent, zum einen aus mangelnder Verfügbarkeit von Analog/Digital-Wandlern mit entsprechenden Sampleraten, zum anderen aufgrund der enormen Anforderungen an die Signalverarbeitung. Es finden sich jedoch zahlreiche Optimierungen und Fallstudien für DDC-Designs im Hundert-Megasample Bereich [61]. Diese Anforderungen an den Durchsatz erlauben eine Standardimplementierung, teilweise konnten Implementierungen erzielt werden welche lediglich IP-Cores von Xilinx verwenden [118].

Ein vollständiges DDC-Design für eine Eingangssamplerate im Gigasample-Bereich ist zwar nicht bekannt, für den komplexesten Teil eines solchen DDC, dem Tiefpassfilter, zeigte Alcatel Lucent allerdings 2010 eine Lösung [24] für bis zu 20 GSa/s. Allerdings benötigte dieses Filter nahezu alle Ressourcen eines sehr großen, modernen FPGA. Das Design transformiert die Eingangsdaten in eine Darstellung im Frequenzbereich, multipliziert die Daten daraufhin mit der Impulsantwort des Filters, um das Ergebnis anschließend wieder in eine Darstellung im Zeitbereich zurück zu transformieren, siehe Abbildung 3.4. Durch diese Art

und Weise lassen sich FIR-Filter von sehr hoher Ordnung wesentlich effizienter realisieren als auf herkömmliche Weise.

4. Flexibles MPSoC zur Hochgeschwindigkeits-OFDM Signalverarbeitung

Dieses Kapitel stellt im Detail die Architektur und Komponenten der Systeme vor, welche mit Hilfe des HW/SW Co-Design Frameworks entstehen sollen. Zunächst wird in Kapitel 4.1 ein Anwendungsszenario eingeführt, welches als Beispiel eines OFDM-Systems dient und an welchem sich die Entwicklung orientiert. Anschließend werden in Kapitel 4.1.2 die Basisfunktionalitäten eines OFDM Systems analysiert, um die Anforderungen an eine für die Verarbeitung von OFDM-Signalen optimierte Architektur zu erarbeiten.

Aufbauend darauf wird in Kapitel 4.2 die Festlegung der allgemeinen Systemeigenschaften diskutiert, um in Kapitel 4.3 und Kapitel 4.4 eine detaillierte Beschreibung des Aufbaus der einzelnen Komponenten zu geben.

Es sei angemerkt, dass es sich bei allen Angaben in diesen Kapiteln zum Ressourcenverbrauch und zur maximalen Taktfrequenz um Ergebnisse des *Xilinx Synthese Tools* (XST) in der Version 14.6 handelt, wobei lediglich die Option für *Register Balancing* genutzt wurde, vergleiche Retiming [59]. Daher sind diese Daten nur zur groben Orientierung gedacht. Genauere Angaben finden sich bei den Ergebnissen komplett implementierter und getesteter Sender und Empfänger in Kapitel 5, dort ist ebenfalls der Ressourcenverbrauch für einzelne Komponenten aufgeschlüsselt.

4.1. Anwendungsszenario und Algorithmenpartitionierung

4.1.1. Vernetzung von Mobilfunkbasisstationen

Obwohl die zu entwickelnde Architektur allgemein für die OFDM-Signalverarbeitung optimiert sein soll, wurde ein Referenzszenario verwendet, anhand dessen die Entwicklung durchgeführt wurde. Das betrachtete Szenario liegt in der Vernetzung von Mobilfunkbasisstationen, welche zukünftig zum Einsatz von fortgeschrittenen Optimierungstechniken, wie beispielsweise Beamforming [22] oder Coordinated Multi-Point (COMP) [54], benötigt werden. Hierdurch wird es mög-

lich, durch das Wissen über die Sendedaten benachbarter Basisstationen Einfluss auf das eigene Signal zu nehmen und damit die Gesamtabdeckung zu optimieren.

Für solche Techniken ist es nötig, die Basisstationen zum Austausch der Sendedaten untereinander zu vernetzen, wobei für diesen Austausch bei COMP Datenraten im einstelligen Gigabit pro Sekunde-Bereich pro Link benötigt werden, siehe [15]. Für gewöhnlich wird ein Punkt-zu-Punkt Netzwerk aufgebaut, jede Basisstation wird je nach Implementierung mit den X-nächsten Nachbarbasisstationen verbunden. Dazu erhält jede Basisstation mindestens einen Sender sowie X-Empfänger.

Eine mögliche Realisierung solch eines Netzwerkes soll mit Hilfe des HW/SW Co-Design Frameworks und einer optischen-Übertragung realisiert werden. Jede Basisstation mischt das mittels eines OFDM-Senders erzeugte Basisbandsignal auf eine stationsspezifische elektrische Trägerfrequenz. Mit dem resultierenden Signal wird dann wiederum ein Laser moduliert, vergleiche Abbildung 4.1.

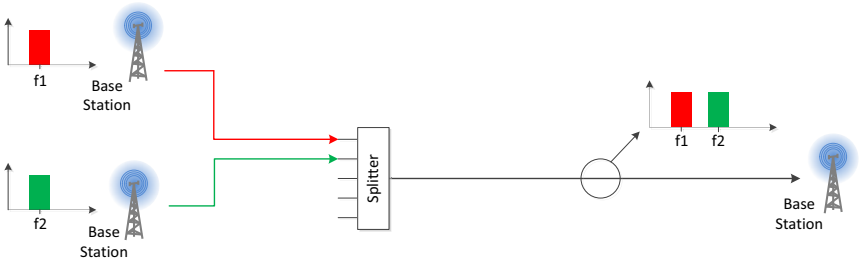


Abbildung 4.1.: Jede Basisstation erhält eine eigene Modulationsfrequenz für den Laser, wodurch die Sendesignale mit Hilfe eines Splitters zusammengelegt werden können und der Empfang mittels einer einzigen Photodiode ermöglicht wird.

Durch den Einsatz von *Splittern* werden die Sendesignale auf eine einzelne Glasfaser gebündelt, um so mit Hilfe dieses Konzeptes anstelle eines separaten Empfängers pro benachbarter Basisstation nur noch einen Empfänger mit nur einer Photodiode zu benötigen, vergleiche Abbildung 4.2.

In einem solchen Empfänger digitalisiert zuerst ein besonders breitbandiger Analog/Digital-Wandler das Empfangssignal, welches die verschiedenen OFDM Signale aller Nachbarstationen enthält. Anschließend wird für jede benachbarte Basisstation durch einen eigenen hochparallelen *Digital Down Converter* (DDC) das OFDM Signal von der entsprechenden Trägerfrequenz ins Basisband gemischt, einer Tiefpassfilterung unterzogen, sowie in seiner Samplerate reduziert. Hierdurch können die einzelnen Signale voneinander getrennt werden. Die nachgeschalte-

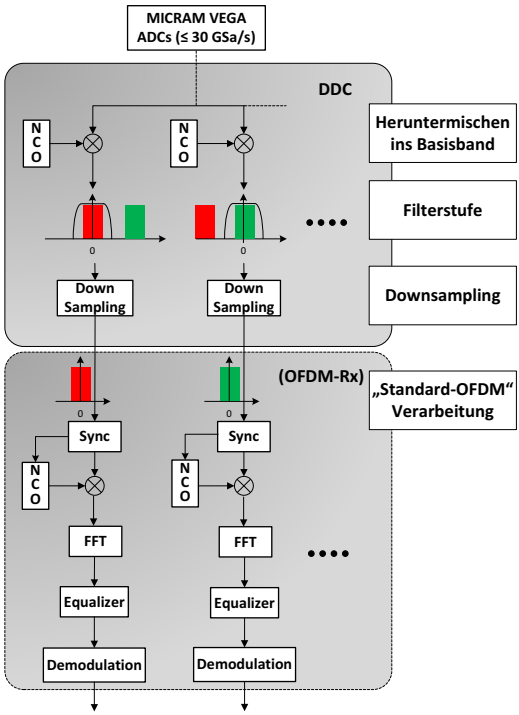


Abbildung 4.2.: Der Empfänger im Anwendungsszenario besitzt eine gemeinsame Optik und einen sehr breitbandigen ADC. Die Signalverarbeitung geschieht für jedes Empfangssignal getrennt durch jeweils einen DDC mit anschließendem OFDM Empfänger.

ten Empfänger, die, wie bereits der Sender, aus den Komponenten des Frameworks zusammengestellt werden können, verarbeiten anschließend die Signale.

4.1.2. HW/SW Co-Design der OFDM-Signalverarbeitungskette

Bevor mit der Entwicklung einer optimierten Architektur begonnen werden kann, sollten zunächst die einzelnen Signalverarbeitungsaufgaben identifiziert werden. Abbildung 4.3 zeigt diese Signalverarbeitungsschritte der physikalischen Schicht eines typischen OFDM basierten drahtlosen Kommunikationssystems.

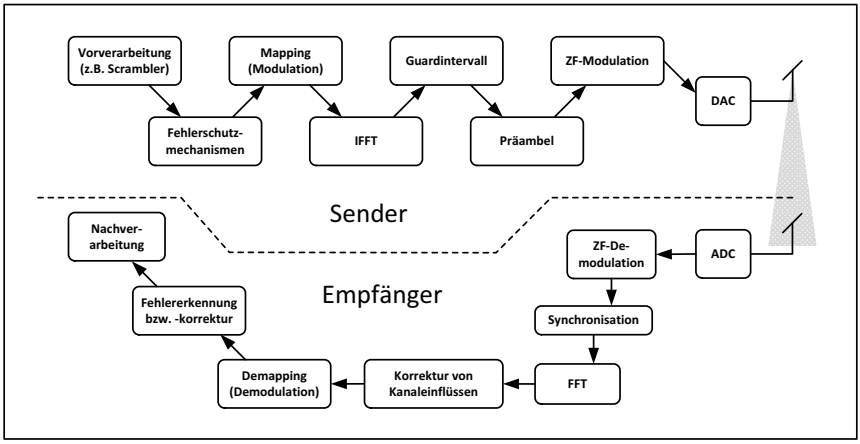


Abbildung 4.3.: Die typischen Signalverarbeitungsschritte der physikalischen Schicht eines drahtlosen OFDM-Systems [77].

Die Komponenten zur digitalen Modulation auf eine Zwischenfrequenz und die entsprechende Komponente zur digitalen Demodulation im Empfänger sind hierbei optional zur Vermeidung von jeweils zwei analogen Wandlern und sollen nicht weiter berücksichtigt werden. Gleiches gilt für Komponenten zur Vorverarbeitung und zum Fehlerschutz. Sie sind für die zuverlässige Übertragung in Produktivsystemen zwar unumgänglich, jedoch nicht OFDM-spezifisch und in vielen Anwendungsfällen in Laboren, wo eine korrekte Übertragung von Daten nicht unbedingt im Mittelpunkt steht, kann oftmals darauf verzichtet werden.

Eine in [Mer13] aufgebaute Simulation eines OFDM-Übertragungssystems bestätigt die Funktionalität und Vollständigkeit dieser Schritte und der involvierten Algorithmen. Die Simulation ist vollständig innerhalb Matlab-LabVIEW aufgebaut und erlaubt das Einstellen und Evaluieren einer Vielzahl von Parametern. Durch den modularen Aufbau können Erweiterungen schnell integriert und untersucht werden.

Bei den identifizierten Komponenten eines Senders handelt es sich somit um die folgenden Module:

- Mapping: Zur Modulation der Subträger, vgl. Kapitel 2.1.1
- IFFT: Zur Erzeugung des Zeitsignals, vgl. Kapitel 2.1.3
- Guardintervall: Zum Schutz vor ISI, vgl. Kapitel 2.1.4
- Präambel: Zur Synchronisation und zum Training des Entzerrers

Entsprechend bestehen die benötigten Funktionalitäten innerhalb eines Empfängers aus:

- Synchronisation: im Zeit- und Frequenzbereich, vgl. Kapitel 2.1.5
- IFFT: Zur Trennung der einzelnen Subträger
- Korrektur von Signaleinflüssen: vgl. Kapitel 2.1.6
- Demapping: Rückgewinnung der Information aus den Symbolen

In Abbildung 4.3 übernimmt das Mapping Modul das Hinzufügen von Pilottönen im Sender, die Entfernung des Guardintervalls wird zur Synchronisierung hinzugezählt. Die effiziente, flexible Abarbeitung dieser Signalverarbeitungskette gilt es zu realisieren. Dabei bilden die einzelnen Aufgaben bereits natürliche Grenzen für eine Realisierung in Software oder mittels Hardwarebeschleuniger. Um allgemeine Designentscheidungen für die Architektur treffen zu können, ist es wichtig abzuschätzen, welchen Anteil die einzelnen Funktionen an der Erstellung des Gesamtsignals besitzen.

Geht man von Hochgeschwindigkeitsübertragungssystemen aus, bei welchen die digitale Signalverarbeitung die limitierende Rolle in Bezug auf das Gesamtsystem spielt, ist offensichtlich, dass das System einen möglichst großen aber auch relativ konstanten Durchsatz bieten sollte. Somit sollten die wichtigsten Funktionen zur Bildung beziehungsweise Verarbeitung des Sende- beziehungsweise Empfangssignals durch eine Verarbeitung mittels Hardware beschleunigt werden.

Da das Mapping Modul die Erstellung der einzelnen Subträger anhand der OFDM Konfiguration und der Sendedaten übernimmt, ist der Durchsatz dieser Funktion ein begrenzender Faktor des Systems und sollte optimiert, also mit Hilfe eines Beschleunigers durchgeführt werden. Das gleiche gilt ohne Einschränkung für die Funktionalität der inversen schnellen Fouriertransformation, da sie aus den Samples für die einzelnen Subträger das Zeitsignal, also das eigentliche Sendesignal bildet.

Das Hinzufügen eines Guard Intervalls wird nur einmal pro OFDM-Symbol benötigt. So kommt, je nach Subträgeranzahl, diese Funktionalität relativ selten vor, sodass sich eine aufwendige Beschleunigung nicht lohnt. Ebenso wird eine Präambel nur einmal zu Beginn eines OFDM-Frames gesendet. Sie ist für jeden Frame identisch, wodurch für die Erstellung der Präambel kein Extra-Beschleuniger verwendet werden sollte.

Im Gegensatz zur Präambel muss die Funktion der Zeit- und Frequenzsynchronisierung mittels Hardware ausgeführt werden. Einerseits werden die Korrelationsfunktionen der Zeitsynchronisierung während des Empfangs eines OFDM-Frames nicht benötigt. Andererseits müssen sie mit dem vollständigen Durchsatz der eingehenden Samples erstellt werden, falls noch keine Zeitsynchronisierung vorliegt. Dies ist durchgehend notwendig, um den Start eines OFDM-Frames si-

cher zu erkennen. Gleiches gilt für die Synchronisierung im Frequenzbereich, dessen Frequenzkorrektur während des Empfangs durchgehend stattfinden muss.

Die Korrektur von Kanaleinflüssen sollte ebenfalls in Hardware geschehen, da sie auf das Sample eines jeden empfangenen modulierten Trägers angewandt werden muss. Die Berechnung der Korrekturfaktoren erfolgt zumindest bei der durch Pilotträger nachgeführten Korrektur durchgehend und benötigt komplexe Divisionen, die sehr aufwendig sind. Deshalb kommt auch diese Funktionalität nicht ohne Beschleuniger aus.

Die Berechnung der Faktoren anhand einer Präambel könnte auch in Software durchgeführt werden. Da sie ebenso wie die durch Pilottöne gestützte Korrektur komplexe Divisionen benötigt, diese Divisionen aber nicht zeitgleich durchgeführt werden müssen, könnten beide Varianten durch einen einzigen Beschleuniger realisiert werden.

Für die FFT und für das Demapping im Empfänger sollten aus den selben Gründen wie für die äquivalenten Funktionen des Senders Beschleuniger verwendet werden.

Um die Architektur selbst zu testen, aber auch um später im Labor mittels Datenübertragung Tests durchzuführen, wird noch eine on-Chip Datenquelle als auch eine on-Chip Datensenke benötigt. Im Sender eignet sich hierzu ein einfacher deterministischer Zufallsgenerator, welcher eine Pseudorandom Binary Sequence (PRBS) erzeugt. Da ein Sender nur senden kann, wenn Sendedaten zur Verfügung stehen, sollte diese Funktionalität ebenfalls in einem Beschleuniger umgesetzt werden. Gleiches gilt für ein entsprechenden PRBS Checker, welcher im Empfänger die selben PRBS Daten erzeugt und mit den empfangenen Daten vergleicht.

Zusammengefasst sollten die Berechnungen der Signalverarbeitungsblöcke, welche in Abbildung 4.4 dargestellt sind, beschleunigt werden, beispielsweise durch Coprozessoren beziehungsweise Hardwarebeschleuniger. Dabei werden die bereits erwähnten Funktionen noch um Schnittellen zur Ein- und Ausgabe ergänzt (DAC IF und ADC IF).

4.2. Systemarchitektur

Die Architektur für das zu entwickelnde System war durch die gesetzten Ziele bereits stark vorgegeben. Während eine dedizierte Schaltung als Architektur aufgrund der geforderten Flexibilität ausscheidet, hat sich in Kapitel 3 gezeigt, dass die Konzepte, welche die Möglichkeit der Programmierung durch Software erlauben, bisher zu wenig Performanz bieten. Da es Ziel der Architektur ist einen Kompromiss zwischen Flexibilität und Performanz zu erreichen, kann man Ab-

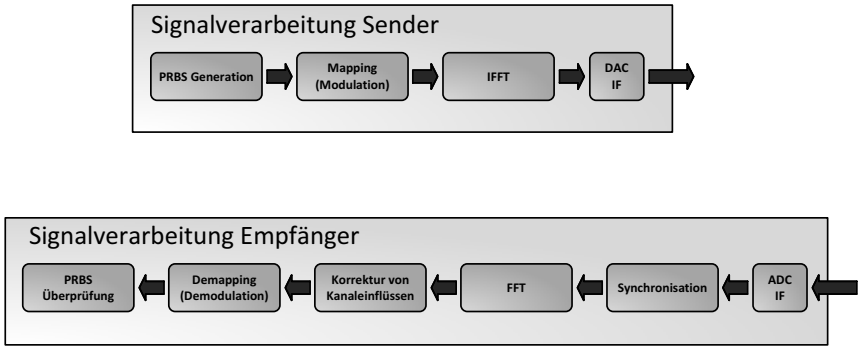


Abbildung 4.4.: Signalverarbeitungsschritte eines Standard OFDM-Systems, welche zur Realisierung durch Hardware Beschleuniger identifiziert wurden.

bildung 2.35 entnehmen, dass sich die angestrebte Architektur an der Grenze von Hardware zu Software befinden muss, also ungefähr dort, wo sich der FPGA befindet.

Ein Standard-Schaltungsentwurf für FPGAs sieht die Realisierung einer dedizierten Schaltung vor. Da dies aber zu unflexibel ist, sollen hochspezialisierte, programmierbare Spezialprozessoren zum Einsatz kommen. Diese sollen so stark spezialisiert sein, dass es sich nicht mehr um Prozessoren im eigentlichen Sinne handelt, sondern um Rechenkerne, auch Hardwarebeschleuniger (Hardware Accelerator) genannt. Diese sind zwar konfigurier- und programmierbar, jedoch nur auf eine Art von Operation ausgelegt.

Im System wird trotz der starken Spezialisierung eine hohe Zahl dieser programmierbaren Hardwarebeschleuniger benötigt, um höchste Datenraten zu erreichen. So können zum einen mehrere OFDM Symbole parallel berechnet und verarbeitet werden, da diese in der Regel keine direkten Abhängigkeiten besitzen. Andererseits kann jeder Verarbeitungsschritt innerhalb der Verarbeitungskette von einem eigenen Beschleuniger durchgeführt werden. So sind im Idealfall alle Beschleuniger durchgehend ausgelastet und nicht durch eine langsame, in Software realisierte Funktion ausgebremst.

Durch das Festlegen der im System befindlichen Beschleuniger ist es möglich, zur Designzeit den Kompromiss zwischen Ressourcen- beziehungsweise Energieverbrauch und maximaler Datenrate zu bestimmen. Dabei ist es wichtig, dass die Architektur zumindest in einem gewissen Umfang skaliert, damit eine Erhöhung

der Anzahl der Beschleuniger entsprechend wenig Einfluss auf den maximalen Systemtakt ausübt.

Neben der Flexibilität zur Designzeit ist auch Flexibilität bezogen auf die OFDM Parameter, welche zur Laufzeit durch Software angepasst werden sollen, ein Ziel dieser Arbeit. Da dies möglichst einfach und ohne größere Vorkenntnisse geschehen sollte, macht es Sinn, die Steuerung und Programmierung der Hardwarebeschleuniger durch ein zentrales System zu verwalten.

4.2.1. MPSoC Konzept

Auf hoher Abstraktionsebene kann das System wie in Abbildung 4.5 gezeigt, in zwei unabhängige Teilsysteme getrennt werden, in ein *Control System* (CS) und in ein *Signal Processing System* (SPS). Das Signal Processing System stellt dabei ein heterogenes Multiprozessorsystem dar, welches aus einer Sammlung von Hardwarebeschleunigern besteht, die auf die Verarbeitung von OFDM Signalen spezialisiert sind. Die Konfiguration und Programmierung der einzelnen Beschleuniger des SPS sowie die Schnittstelle zum Benutzer wird durch das *Control System* realisiert. Somit lässt sich das Konzept grob mit dem LME2980 OFDM Chip aus Abbildung 3.1 vergleichen, wobei das SPS den OFDM Transceiver des LME2980 darstellt, das CS den Rest des MPSoCs.

Da das SPS und das CS jeweils einen eigenen, unabhängigen Takt verwenden, müssen alle Verbindungen zwischen den Systemen entsprechend synchronisiert werden. Dies hat den Vorteil, dass die maximale Taktfrequenz des Signalverarbeitungssystems nicht von der maximalen Taktfrequenz des Kontrollsystems abhängig ist. Dadurch können die Beschleuniger des SPS auch im Falle eines komplexen und eventuell langsam getakteten Prozessors des CS ausgereizt werden.

Um höchste Datenraten zu erreichen, soll idealerweise die gesamte Signalverarbeitung für das System im SPS stattfinden. Allerdings sollte es dennoch möglich sein, durch den CS-Prozessor per Software in die Signalverarbeitung einzugreifen, um beispielsweise neue Algorithmen oder Verarbeitungsschritte zu evaluieren.

4.2.2. Arbeitsspeicher

Die Speicherarchitektur stellt eine wesentliche Komponente des Systems dar. Damit der Arbeitsspeicher nicht als Flaschenhals das System ausbremst, muss die Speicherbandbreite so groß sein, dass alle vorhandenen Hardwarebeschleuniger in jedem Taktzyklus genügend Daten lesen und schreiben können. Dies ist mit den in aktuellen Systemen standardmäßig verwendeten Speichern wie beispiels-

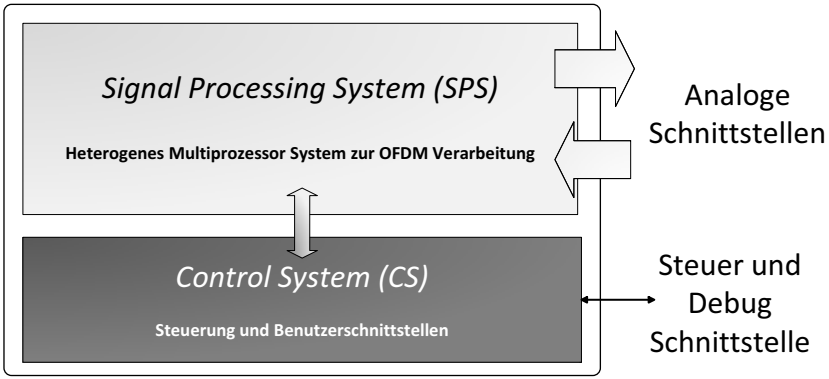


Abbildung 4.5.: Unterteilung des Systems in das Signal Processing System zur Signalverarbeitung und das Control System für Kontroll- und Steueraufgaben.

weise DDR3-SDRAM nur äußerst schwer zu erreichen. Soll solch ein System noch skalierbar sein, wird dies für die anvisierten Durchsatzraten nahezu unmöglich.

Für die Zielarchitektur, FPGAs der Virtex-6 Familie, besteht neben der Möglichkeit einen externen Speicher anzubinden auch die Option, die on-Chip Speicher Primitive zu verwenden. Für das SPS wird als Arbeitsspeicher eine on-Chip Realisierung durch solche BRAM Elemente verwendet. Der Vorteil der BRAM Primitive ist, dass die Verwendung einfach ist, sie äußerst schnell sind und sie, falls genügend BRAM Primitive vorhanden sind, leicht und flexibel zusammengeschaltet werden können, um tiefere Speicher oder größere Wortbreiten zu realisieren.

Da es sich bei BRAM um Dual-Port Speicher (DP-BRAM) handelt, diese also über zwei vollwertige Schnittstellen verfügen, lassen sich auf solche Speicher gleichzeitig Schreib- als auch Lese-Operationen ausführen. Anstelle eines komplexen Speichercontrollers können leicht mehrere solcher Speicher verwendet werden, wodurch die Speicherarchitektur - genügend Ressourcen vorausgesetzt - auch gut skaliert. Der einzige Nachteil liegt in der beschränkten Anzahl an BRAM Elementen.

4.2.3. Zahlendarstellung

Eine sehr wichtige Rolle für die Genauigkeit sowie den Ressourcenverbrauch des Systems spielt seine Wortbreite. Sie darf weder zu klein sein, da sonst eventuell auftretende Rechenfehler durch Runden der Ergebnisse zu groß werden und

sich immer weiter fortpflanzen können, noch darf sie zu groß sein, da sich der Ressourcenverbrauch mit jedem Bit drastisch erhöht. Im Laufe der Zeit hat sich ein Byte als kleinste adressierbare Zelle durchgesetzt. Daher sollte die Wortbreite auch in 8-Bit unterteilbar sein, um mit Standard Bussystemen und anderen Komponenten leichter kompatibel zu sein.

Einen wichtigen Hinweis über die benötigte Genauigkeit gibt die Auflösung der zum Einsatz kommenden Analog/Digital- beziehungsweise Digital/Analog-Umsetzer. Gerade im absoluten Hochgeschwindigkeits-Bereich existieren wenige Umsetzer mit hoher Auflösung. Beispielsweise bietet die Firma Texas Instruments zur Zeit dieser Arbeit Analog/Digital-Wandler mit Sampleraten im Bereich von 10 MSa/s bis 250 MSa/s und mit einer Auflösung von 16-Bit an. Bis 400 MSa/s werden entsprechende Wandler mit 14-Bit Auflösung angeboten, darüber (bis 3,6 GSa/s) sinkt die Auflösung auf 12-Bit.

Ein anderes Beispiel sind die Analog/Digital-Wandler *VEGA ADC 30* der Firma Micram, welche zwar Sampleraten bis zu 30 GSa/s liefern können, allerdings auch nur eine Auflösung von 6-Bit besitzen. Bedenkt man, dass sich die effektive Anzahl an Bits eines Wandlers noch einmal durch Verzerrungen und Rauschen verringert, ist ersichtlich, dass eine gute Wortbreite zwischen 16-Bit und 24-Bit liegt.

Die realisierte Architektur verwendet eine Wortbreite von 16-Bit, da diese Wortbreite wesentlich üblicher ist. So besteht ein komplexes Sample aus zwei 16-Bit Zahlen, welche sich im Gegensatz zu zwei 24-Bit Zahlen gut in einem einzigen BRAM speichern lassen. Allerdings ist es durch den konsequenten Einsatzes von VHDL-Generics verhältnismäßig einfach, die Wortbreite der Architektur zu erhöhen, sollte es zukünftig erforderlich werden.

4.2.4. Verbindungsstruktur

Ganz entscheidend für eine leistungsfähige Architektur zur Signalverarbeitung ist die Verbindungsstruktur. Gerade bei Multiprozessorsystemen muss sie ein Vielfaches der Eingangs- beziehungsweise Ausgangsdatenrate unterstützen. So sollte die maximale Datenrate der Verbindungsstruktur ausreichen, um jederzeit genügend Eingangsdaten zu den Rechenkernen zu bringen, sowie um die Ergebnisse der Rechenkerne in den Arbeitsspeicher zurück zu transferieren. Gerade in der Kommunikationstechnik, welche meist durchgehend einen konstanten Strom an Daten verarbeitet, ist Verbindungsstruktur unbedingt für eine entsprechenden Durchsatz auszulegen.

Neben dem reinen Durchsatz können allerdings auch noch andere Eigenschaften von Bedeutung sein. So spielt die Skalierbarkeit für eine flexible MPSoC Architektur eine wichtige Rolle, zum einen in Bezug auf den Ressourcenverbrauch, zum

anderen in Bezug auf die Datenrate bzw. auf die maximale Taktfrequenz. Die Latenz der Verbindung ist hingegen bei solchen Kommunikationssystemen meist weniger kritisch. Vor allem zu Gunsten eines höheren maximalen Taktes kann eine definierte, konstante Latenz in einem gewissen Rahmen durchaus tolerierbar sein.

Im Wesentlichen existieren vier bis fünf Architekturen für Verbindungsstrukturen, *Punkt-zu-Punkt*, *single und hierarchical shared bus*, *Multilayer Busmatrix* auch *cross bar switch* genannt, sowie das Konzept eines *Network-on-Chip* (NOC), vergleiche [72] sowie [53]. Aufgrund der hohen Anforderungen an den Datendurchsatz kommen für diese Arbeit nur Verbindungsstrukturen in Frage, welche für hohe Datenraten ausgelegt sind, also eine Punkt-zu-Punkt Verdrahtung, eine Multilayer Busmatrix oder ein Network-on-Chip. Ein shared bus, ein Systembus der von allen geteilt wird und mehrere parallele Datentransfers nicht erlaubt, ist ungeeignet, um in einem MPSoC alle Hardwarebeschleuniger auszulasten. Dazu müssen diese prinzipiell alle zur selben Zeit und durchgehend auf den Bus zugreifen können, um neue Daten zu lesen und eventuell sogar gleichzeitig neue Ergebnisse zu speichern.

Das Konzept einer Punkt-zu-Punkt Verdrahtung ist für viele Fälle der OFDM Signalverarbeitung ausreichend, da sich der Ablauf der Signalverarbeitung in der Regel nicht ändert. Die Verdrahtung kann daher direkt den Datenfluss abbilden und erzielt höchste Durchsatzraten, da Datentransfers immer stattfinden können, vergleiche beispielsweise die Datenflußstruktur eines Wishbone Busses [71]. Solch eine Struktur wäre somit die effizienteste Lösung, für die Ziele dieser Arbeit ist sie jedoch zu unflexibel.

Eine Multilayer (Bus-) Matrix ist in der Lage, viele parallele Datentransfers gleichzeitig zu unterstützen. Dadurch erhöht sich der Durchsatz einer solchen Struktur drastisch im Vergleich zu geteilten Bussystemen, was eine Multilayer Matrix, wie auch die Punkt-zu-Punkt Verbindung, zu einer der durchsatzstärksten Verbindungsstrukturen macht. Nachteile sind der relativ hohe Ressourcenverbrauch und auch die eher schlechten Eigenschaften in Bezug auf Skalierbarkeit.

Network-on-Chips stellen das jüngste Konzept einer Verbindungsstruktur dar. Es kann, je nach Implementierung, hohe Datenraten und auch eine gute Skalierbarkeit bei angemessenem Ressourcenverbrauch bieten. Die Entwicklungszeit eines solchen Netzwerks ist allerdings nicht zu unterschätzen. Gerade wenn noch Nebenbedingungen an die maximale Datenrate und die Latenzen einer Verbindung bestehen, wird für ein NOC eine recht komplexe Flusskontrolle benötigt, welche in einem Kommunikationssystem mit festem Datenfluss den Großteil der Betriebszeit überflüssig ist. Des Weiteren sind NOCs immer noch ein großer Forschungsschwerpunkt. Um zu aktuellen Komponenten und Modulen kompatibel zu sein, eignen sich normale Bussysteme oder eine Busmatrix meist besser.

Obwohl ein entsprechend ausgelegtes Network-on-Chip Vorteile bieten dürfte, verwendet diese Arbeit aufgrund der höheren Entwicklungszeit von NOCs und aufgrund eines besser vorhersehbaren Datenflusses eine Multilayer Busmatrix als Verbindungsstruktur. Als Besonderheit lässt sich diese auch zur Designzeit zu einer Punkt-zu-Punkt Verdrahtung konfigurieren, wodurch es möglich ist, sich für mehr Flexibilität oder eine höhere Ressourceneffizienz zu entscheiden. Dabei ist in beiden Fällen gewährleistet, dass jeder Master des Busses jederzeit Zugriff auf den Bus erhalten kann.

Um möglichst kompatibel zu den Modulen und Systemen Dritter zu sein, wurden die Verbindungsstrukturen nach einem verbreiteten Industriestandard, den AMBA-2 Spezifikationen der Firma ARM realisiert, siehe [5]. Vorteile gegenüber konkurrierenden Systemen wie Wishbone oder dem CoreConnect Bus-System, sowie gegenüber Weiterentwicklungen wie AXI sind eine sehr weite Verbreitung, beispielsweise durch die Aeroflex Gaisler IP-Core Bibliothek GRLIB [3], sowie eine sehr gute Dokumentation.

Konkret wurde für das SPS eine 64-Bit breite AHB Multilayer Matrix entwickelt, im CS kommt ein 32-Bit AHB System der Firma Aeroflex Gaisler zum Einsatz. Ein entsprechendes Blockschaltbild des Systems ist in Abbildung 4.6 zu sehen. Einzelheiten zum CS oder zur Multilayer Matrix folgen in Kapitel 4.3.1 beziehungsweise in Kapitel 4.4.1.

4.2.5. Flexibilität zur Designzeit

Zur Designzeit des Systems ist es möglich, dessen Eigenschaften in bestimmte Richtungen zu optimieren. Die zentrale Frage ist hierbei der Kompromiss zwischen maximaler Datenrate und dem Ressourcenverbrauch, welcher für Kosten und Energieverbrauch steht. Hauptziel der Architektur ist es, ein flexibles System auf Durchsatz zu optimieren.

Zur Optimierung auf Leistung ist es essentiell, alle Schritte der Signalverarbeitung in entsprechenden Beschleunigern zu realisieren. Ein Einbeziehen der CPU des CS in die Signalverarbeitung würde zum Flaschenhals des Systems werden. Um das System noch weiter auf Leistung zu trimmen, ist es sinnvoll, das SPS des Designs in mehrere unabhängige Signalverarbeitungspfade aufzuteilen, welche im folgenden *Stream* oder SPS-Stream genannt werden.

Jeder Stream bearbeitet ein unabhängiges OFDM Symbol. Die analoge Schnittstelle hat im Falle mehrerer Streams die Aufgabe, Samples in Symbole zu gruppieren. Im Falle eines Senders werden die Daten von allen Streams entgegen genommen und in korrekter Reihenfolge an die Digital/Analog-Umsetzer weiter gegeben. Im Falle eines Empfängers muss die analoge Schnittstelle die eingehenden OFDM

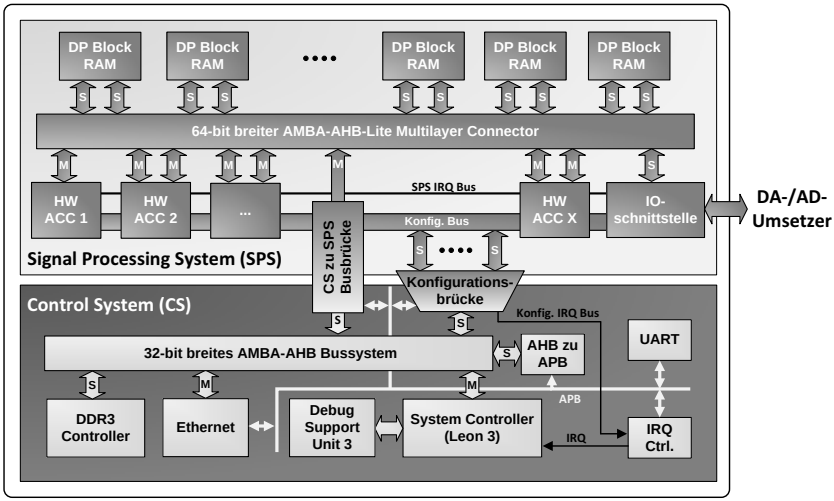


Abbildung 4.6.: Um genügend Datendurchsatz zu bieten kommt im SPS eine Multilayer Matrix sowie eine Speicherstruktur mit mehreren Kanälen zum Einsatz.

Symbole auftrennen und korrekt den Streams zuführen. In Abbildung 4.7 ist das Prinzip an einer Empfängerimplementierung mit 3 Streams abgebildet.

Definition 4.1 (Stream) *Ein (SPS-) Stream bezeichnet einen unabhängigen Signalverarbeitungspfad innerhalb des Signal Processing Systems.*

Trotz der Ausrichtung auf eine maximale Datenrate ist es durchaus möglich, mit Hilfe des Frameworks ein energieeffizientes System aufzubauen, welches zwar keine hohen Durchsatzraten liefert, dafür jedoch nur einen geringen Ressourcenverbrauch aufweist. So kann ein System beispielsweise, in Anlehnung an die Espresso-Plattform (vgl. Abbildung 3.3), nur aus dem LEON-3 Subsystem, einem Synchronisationsbeschleuniger, einem (i)FFT-Beschleuniger und der Schnittstelle zum ADC bestehen. Ein entsprechendes Design wurde beispielsweise in Kapitel 7.2 für die energiesparende und kostengünstige Spartan- Familie erstellt.

Zu Beginn sollte zudem erwogen werden, ob ein Sender, ein Empfänger oder ein Sende-Empfänger (Transceiver) implementiert werden soll. Im Falle eines Transceivers ist zu entscheiden, ob das Design ein gleichzeitiges Senden und Empfangen unterstützen muss, oder ob jeweils nur eine Funktion benötigt wird. Dies hat Einfluss auf die Anzahl an Beschleunigern, da bestimmte Algorithmen für Sen-

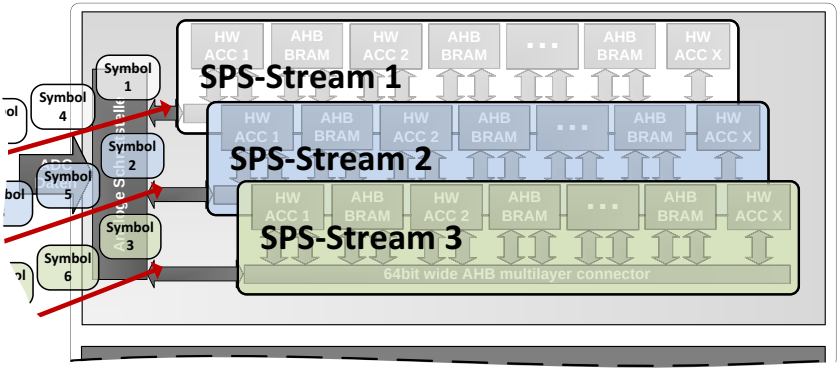


Abbildung 4.7.: Die Unterteilung des SPS in mehrere Streams ermöglicht den Einsatz besonders vieler ASIPs, trotz der eingeschränkten Fähigkeit der Busmatrix zu skalieren.

der als auch für Empfänger eingesetzt werden können und die entsprechenden Beschleuniger daher eventuell nicht mehrfach benötigt werden.

Weitere Informationen der Optionen zur Designzeit finden sich in Kapitel 7.

4.3. Das *Control System* (CS) und allgemeine Komponenten

Das CS wird lediglich zur Steuerung und zu Debuggingzwecken benötigt und kann daher durch ein beliebiges Prozessor-Subsystem realisiert werden. Einzige Bedingung ist, dass die 32-Bit Konfigurationsschnittstelle der SPS Beschleuniger, welche als AHB-lite Slave entwickelt wurde und in Kapitel 4.3.3 genauer erläutert wird, angesprochen werden kann. Auch die CS zu SPS Brücke, welche in Kapitel 4.3.2 detailliert vorgestellt wird, folgt den AMBA-lite Spezifikationen [5]. Abbildung 4.8 zeigt das CS dieser Arbeit, inklusive der Busbrücke und der Konfigurationsschnittstelle als Blockschaltbild.

4.3.1. Das *Control System*

In den hier vorgestellten Realisierungen des CS wurden Komponenten der *GRLIB* Komponenten Bibliothek verwendet. Diese Bibliothek stellt fertig getestete Hardwarebeschreibungen, sogenannte *intellectual property cores* (IP-Cores) rund

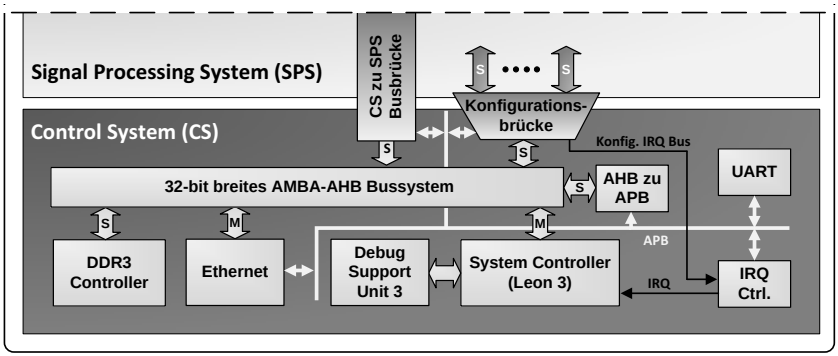


Abbildung 4.8.: Das *Control System* wurde in allen Realisierungen mit Komponenten der *GRLIB* Bibliothek von Aeroflex Gaisler und zusätzlich mit einer Busbrücke sowie einer Konfigurationsschnittstelle aufgebaut.

um den *Leon-3*, einem Sparc-V8 kompatiblen Prozessor, zur Verfügung. Die verwendete Version der GRLIB steht unter der *GNU General Public License*, stammt von der Firma Aeroflex Gaisler und darf zu nicht-kommerziellen Zwecken kostenlos verwendet und verändert werden. Ein großer Vorteil dieses Systems ist das existierende Eclipse Plugin. Dieses bietet in Kombination mit dem Cross-Compiler welchen es beinhaltet und den entsprechenden Debugging IP-Cores der Bibliothek eine vollständige moderne Entwicklungsumgebung für einen geordneten Entwurf und ein komfortables Debuggen von Software für den Leon-3 Prozessor.

Tabelle 4.1.: Syntheseergebnisse des CS für die Virtex-6 Architektur.

Ressourcentyp				Maximale
LUTs	**Flip Flops**	**BRAMs**	**DSP48**	**Taktfrequenz**
21 023	13 991	22	4	294 MHz

Wie aus Abbildung 4.8 ersichtlich, befinden sich neben dem Leon-3 Prozessor und dem dazugehörigen Modul für das Debugging (Debug Support Unit 3, DSU3) auch ein Speichercontroller um DDR3-SDRAM als Arbeitsspeicher anzusprechen, sowie ein Ethernet Controller und eine universelle asynchrone Sende- und Empfangseinheit (Universal Asynchronous Receiver Transmitter, UART) zur Kommunikation. Der Ethernet Controller enthält eine Funktionalität namens *Ethernet Debug Communication Link*, welche ein komfortables und schnelles Debuggen über

Ethernet ermöglicht. Die UART-Schnittstelle ermöglicht eine einfache Programmsteuerung über eine RS232-Schnittstelle. Zur Unterstützung mehrerer Interruptquellen befindet sich noch ein Interrupt Controller im CS, welcher zusammen mit der UART über einen *Advanced Peripheral Bus* (APB) mittels einer AHB zu APB Brücke an das AHB Bussystem angeschlossen ist. Der Ressourcenverbrauch des CS ist in Tabelle 4.1 aufgelistet.

4.3.2. CS zu SPS Brücke

Prinzipiell sind das SPS und das CS zwei voneinander getrennte Subsysteme, welche lediglich durch die Konfigurationsbrücke, siehe Kapitel 4.3.3, sowie der CS zu SPS Brücke miteinander verbunden sind. Letztere dient Situationen, in denen es notwendig ist oder hilfreich sein kann, Daten zwischen den zwei Systemen auszutauschen. So können die Beschleuniger des SPS, gesteuert vom CS System Controller, mit bekannten Eingangsdaten versorgt und deren Ergebnisse auch wieder überprüft werden, um die Beschleuniger auf Funktionalität zu testen oder beispielsweise um Trainingsdaten zu erzeugen. Alternativ kann, im Sinne eines HW/SW Co-Designs, mit Hilfe der Busbrücke der Prozessor des CS zusätzlich an der OFDM Signalverarbeitung mitwirken. Für höchste Datenraten sollte dies jedoch vermieden werden.

Die CS zu SPS Brücke hat sowohl die Aufgabe, dem CS Daten aus dem SPS zugänglich zu machen, als auch Daten in den Speicherbereich des SPS zu schreiben. Dabei ist zu beachten, dass das Bussystem des CS und das Bussystem des SPS nicht nur unterschiedliche Wortbreiten verwenden, sondern auch mit einem voneinander unabhängigen Takt betrieben werden. Weiterhin ist zu bedenken, dass mit der Busbrücke in der Regel ganze Datensätze, meist in der Größenordnung ganzer OFDM Symbole kopiert werden.

Abbildung 4.9 zeigt ein Blockschaltbild der CS zu SPS Brücke. Das Funktionsprinzip basiert auf der Verwendung eines Pufferspeichers. Dieser Puffer- oder Brückenspeicher selbst ist, abhängig von der Tiefe des Speichers, aus mindestens zwei BRAMs aufgebaut. Der Speicher besitzt zwei unabhängige Schnittstellen, von welchen die erste dazu genutzt wird, den Brückenspeicher direkt aus dem CS als 32-Bit AHB Slave Schnittstelle anzusprechen. Über die zweite Schnittstelle kann ein Zustandsautomat mit Hilfe einer 64-Bit Master AHB Schnittstelle Daten vom SPS in den Brückenspeicher schreiben oder auch Daten vom Brückenspeicher in einen Speicherbereich des SPS kopieren. Dieser endliche Automat wird von Seite des CS mittels synchronisierten Steuersignalen aus einem APB Register gesteuert.

Der Vorteil dieser Art von Realisierung liegt in den Burst-Zugriffen, welche das Puffern der Daten ermöglichen. Die Zeit, in welcher ein Slave des SPS durch Zu-

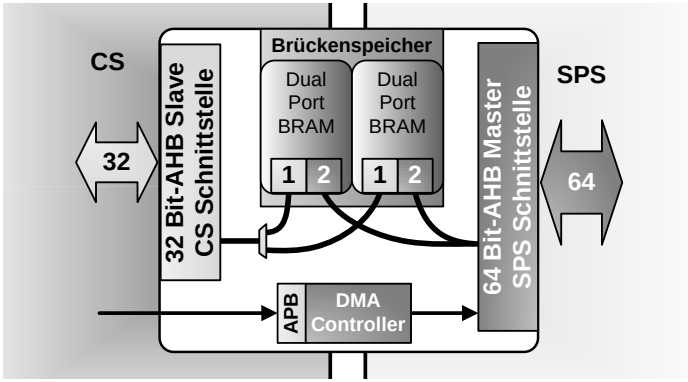


Abbildung 4.9.: Die CS zu SPS Brücke realisiert einen Brückenspeicher, welcher von der CS-Seite als normaler Speicher über AHB angesprochen wird. Über DMA-Zugriffe, welche per APB von der CS Seite aus gesteuert werden, kann die Brücke Daten des Brückenspeichers mit Daten im Speicherbereich des SPS austauschen

Tabelle 4.2.: Syntheseergebnisse der CS zu SPS Brücke für Virtex-6.

Ressourcentyp				Maximale
LUTs	**Flip Flops**	**BRAMs**	**DSP48**	**Taktfrequenz**
653	477	2	0	385 MHz

griffe blockiert ist, kann hierdurch erheblich gegenüber einer direkten Umsetzung der Schreib- und Lese-Zugriffe minimiert werden, insbesondere da der Datenbus des SPS doppelt so breit ist und die Systemfrequenz des SPS nach Möglichkeit auch höher ist als die des CS.

Der Nachteil dieses Konzeptes ist der Verlust der Transparenz, die eine Busbrücke standardmäßig bietet. Ebenso können je nach Verwendung Kohärenzprobleme zwischen dem Pufferspeicher und dem SPS auftreten. Tabelle 4.2 zeigt den Ressourcenverbrauch der CS zu SPS Busbrücke für einen 64 Kb großen Pufferspeicher. Die Speichergröße ist leicht mit der Hilfe von VHDL-Generics zur Designzeit einstellbar.

Falls mehrere SPS-Streams unterstützt werden sollen, kann ein optionaler Busmultiplexer mit 64-Bit Wortbreite hinter die CS zu SPS Brücke platziert und mit Hilfe von APB-Registerbits gesteuert werden. Auf diese Weise ist der Zugriff mittels Busbrücke auf jeden der unabhängigen SPS-Streams möglich.

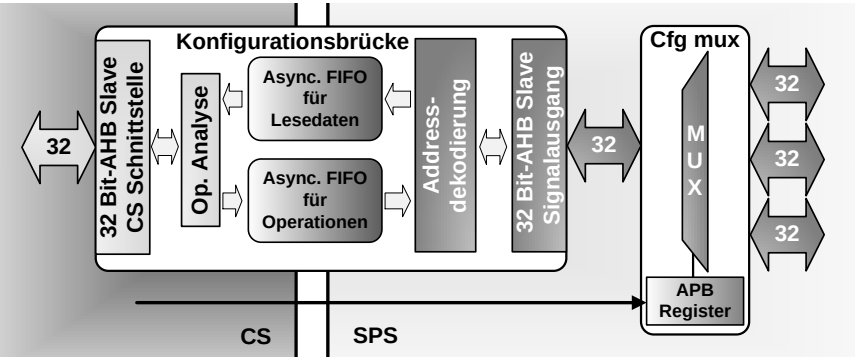


Abbildung 4.10.: Die Konfigurationsbrücke führt ein Clock Domain Crossing der Bussignale durch. Außerdem enthält sie einen Dekoder, um die Adressen der Beschleuniger im SPS richtig aufzulösen. Ein optionaler, per APB gesteuerter Konfigurationsmultiplexer kann anschließend verwendet werden um den korrekten Stream auszuwählen, falls sich mehrere SPS-Streams im Design befinden.

4.3.3. Konfigurationsbrücke

Zur Konfiguration und Programmierung der SPS-Beschleuniger bieten diese eine 32-Bit weite AHB Slave Schnittstelle an. Da das CS und das SPS nicht mit dem gleichen Takt betrieben werden, kann man die Konfigurationsports der Hardwarebeschleuniger allerdings nicht direkt an den Bus des CS anschließen. Auch könnte die potentiell hohe Anzahl an Beschleunigern und damit die hohe Anzahl der Konfigurationsschnittstellen unter Umständen die zulässige Anzahl an Slave Busteilnehmern des CS-Systems übersteigen oder aber unnötig hohe Anforderungen an die Verdrahtung stellen.

Zur Lösung dieser Probleme ermöglicht die Konfigurationsbrücke zum einen eine sichere Übertragung der Daten über die Taktgrenzen der Systeme hinweg (Clock Domain Crossing). Zum anderen führt sie auch ein Multiplexing durch, damit die Konfigurationsschnittstellen des SPS insgesamt nur eine Verbindung zum CS benötigen.

Für das Clock Domain Crossing verwendet die Konfigurationsbrücke zwei BRAM, welche als asynchrone FIFOs konfiguriert sind. Der erste wird verwendet, um die Operation mit den benötigten zusätzlichen Daten auf die SPS Taktdomäne zu bringen, der zweite leitet im Falle von Leseoperationen die Lesedaten zum CS zurück.

Auf CS-Seite stellt die Konfigurationsbrücke eine gewöhnliche Slave Schnittstelle dar, welche den gesamten Adressbereich der Konfigurations- und Programmspeicher aller Beschleuniger des SPS abdeckt. So befindet sich auf SPS-Seite der Brücke ein Adressdekoder, der anhand der Adresse den richtigen Beschleuniger auswählt, das entsprechende Select Signal im Bus setzt und somit ermöglicht, den Konfigurationsbus des SPS daran anzuschließen.

Optional kann hinter die Konfigurationsbrücke noch ein Konfigurationsmultiplexer gesetzt werden. Dieser wird über synchronisierte APB Registerbits gesteuert und ermöglicht es, den Konfigurationsbus für mehrere SPS-Streams (siehe Kapitel 4.2.5) zu verwenden, ähnlich wie bereits für die CS zu SPS Brücke. Tabelle 4.3 zeigt den Ressourcenverbrauch der Konfigurationsbrücke für eine Bereitstellung von fünf SPS-Konfigurationsports sowie den Ressourcenverbrauch des Konfigurationsmultiplexers für 4 SPS-Streams.

Tabelle 4.3.: Virtex-6 Syntheseergebnisse der Konfigurationskomponenten für 5 Slaves und 4 SPS-Streams. Es werden keine DSP48E1 Primitive benötigt.

Komponente	Ressourcentyp			Maximale
	LUTs	Flip Flops	BRAMs	Taktfrequenz
Konfigurationsbrücke (x5)	333	345	2	492 MHz
Konfigurationsmultiplexer (x4)	1031	62	0	1525 MHz

4.4. Komponenten und Beschleuniger des *Signal Processing Systems* (SPS)

Für eine möglichst leistungsfähige Realisierung des SPS sollen alle Beschleuniger durchgehend Berechnungen durchführen können. Die Signalverarbeitung von OFDM Symbolen erfolgt schrittweise in einer Verarbeitungskette. Daher ist es wichtig, dass alle Beschleuniger dieser Kette den gleichen konstanten Datendurchsatz besitzen, damit keiner dieser Rechenkerne einen Flaschenhals bildet und die Datenrate der ganzen Verarbeitungskette senkt. Eine konsequente Anwendung des Pipelining-Prinzips, siehe [70], wird beim Entwurf aller Hardwarebeschleuniger angewandt, sodass alle in jedem Taktzyklus neue Daten entgegen nehmen können.

Da mit einem Buszugriff zwei komplexe Samples transferiert werden können, ist jeder der Beschleuniger des SPS auf einen Durchsatz von zwei Samples pro Taktzyklus optimiert. Somit liegt die Leistungsfähigkeit des Systems bei einer einfa-

chen Implementierung bei ungefähr zwei Samples pro Taktzyklus, bei der Realisierung von mehreren Streams ist der Durchsatz entsprechend gegeben durch

$$\text{Maximaler Systemdurchsatz} = \#\text{Streams} \cdot 2\,\text{Samples/Taktzyklus}. \tag{4.1}$$

Um diesen Durchsatz zu erreichen arbeiten die Beschleuniger nicht nur nach dem Pipelining Prinzip, sie besitzen meist außerdem zwei Anbindungen an den Systembus. Mit der ersten Verbindung werden die zu verarbeitenden Daten aus dem Quellspeicher gelesen und der Verarbeitungspipeline übergeben. Gleichzeitig können mit der zweiten Verbindung die Ergebnisse der Berechnungen in den Zielspeicher geschrieben werden.

Alle Beschleuniger des SPS, auch der Data Output Converter und der Synchronisationsbeschleuniger, bieten eine 32-Bit AHB Slave Schnittstelle zum Zugriff auf ihren Konfigurations- und Programmspeicher. Ebenso bieten alle Beschleuniger die Möglichkeit, einen Interrupt auf dem CS oder dem SPS IRQ Bus zu erzeugen, sowie auf einen Interrupt des SPS Busses zu reagieren. Dies vereinfacht die Einhaltung von Präzedenzrelationen durch die Verarbeitungskette hinweg, da Beschleuniger auf einen Interrupt des Vorgängerbeschleunigers warten können, bevor sie ihr Programm starten.

4.4.1. 64-Bit Multilayer Connector

Der entwickelte AHB Multilayer Connector kann in zwei Versionen implementiert werden. Im Matrix Modus ermöglicht er jedem angeschlossenen Master jederzeit auf jeden beliebigen angeschlossenen Slave zuzugreifen, solange dieser nicht durch den Zugriff eines anderen Masters belegt ist. Die Matrix realisiert Verbindungen mit der Hilfe von Multiplexern, ein entsprechendes Blockschaltbild ist in Abbildung 4.11 zu sehen.

Für eine Datenübertragung ermittelt die Dekodierstufe des Masters, der einen Zugriff einleitet, zuerst den zu kontaktierenden Slave und signalisiert dem Master zu warten. Die Arbitrierungsstufe erkennt die Anfrage der Dekodierung, gibt eine Rückmeldung über den Status des Slave und schaltet, falls dieser frei ist, die entsprechenden Multiplexer. In diesem Fall schaltet die Dekodierung die Mastersignale entsprechend durch und die Datenübertragung beginnt. Ist der Slave nicht frei, muss der Master warten.

Der Vorgang zur Verbindungsherstellung verwendet das Pipelining-Prinzip und benötigt 3 Taktzyklen. Dies ermöglicht es, die Matrix mit einer hohen Frequenz zu betreiben. Die Latenz betrifft nur den Aufbau der Verbindung. Wurde eine Verbindung für einen Burst hergestellt, wird anschließend pro Taktzyklus auch ein Datenwort übertragen.

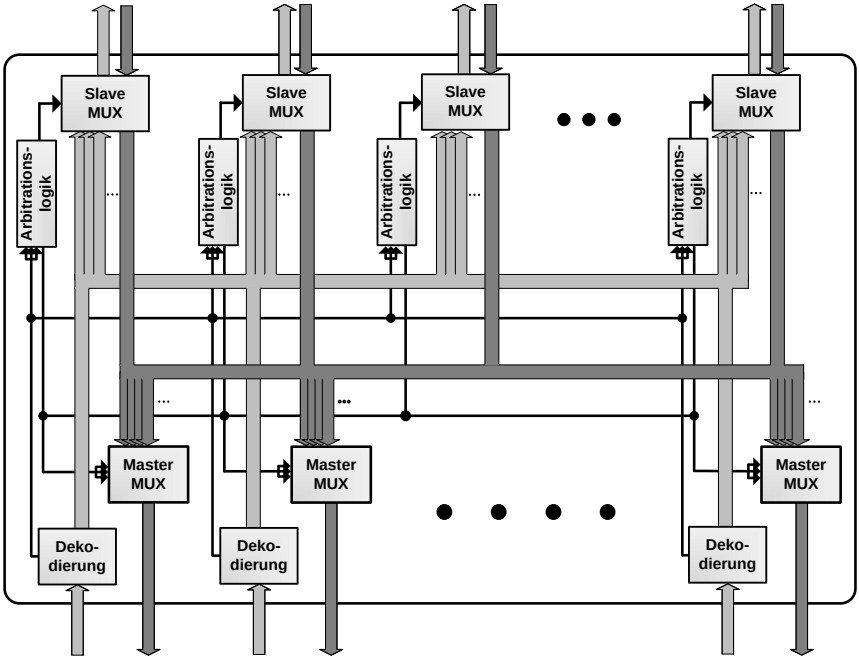


Abbildung 4.11.: Die 64-Bit breite AHB Multilayer Matrix stellt die Verbindung durch Multiplexer her und besitzt eine Arbitrierungslatenz von 3 Taktzyklen. Hierdurch kann sie mit einer hohen Taktrate betrieben werden.

Die Matrix ist, abgesehen von der Latenz des Verbindungsaufbaus, nach den in [6] und [7] beschriebenen Konzepten implementiert. Sie besitzt eine Adressbusbreite von 32 Bit und eine Datenwortbreite von 64 Bit, wobei sich alle Werte leicht über VHDL-Generics anpassen lassen. Die Anzahl der verfügbaren Master und Slave Schnittstellen ist ebenfalls über VHDL-Generics einstellbar, die Arbitrierung verwendet ein festes, aufsteigendes Prioritätsschema.

Die zweite Betriebsart des Connectors bildet der Punkt-zu-Punkt Modus (P2P-Modus). Dieser Modus realisiert eine Punkt-zu-Punkt Verbindungsstruktur, ebenfalls auf AHB-Lite Basis. In diesem Modus hat das Interconnet Modul immer noch die selben Schnittstellen, allerdings ist nur eine gleiche Anzahl an Master und an Slave Schnittstellen zulässig. Im Inneren des Interconnect Moduls wird jede

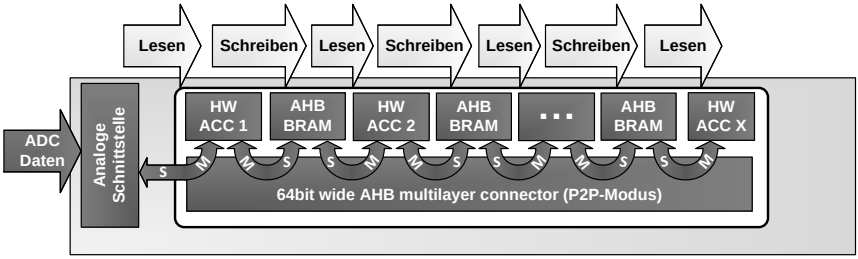


Abbildung 4.12.: Datenfluss im SPS eines Empfängers mit Connector im P2P-Modus. Durch die zwei Schnittstellen eines AHB Speichers ist das Weiterreichen der Daten möglich.

Master Schnittstelle mit ihrer entsprechenden Slave Schnittstelle direkt Punkt-zu-Punkt verbunden.

Der P2P-Modus verbraucht nur wenige Ressourcen, da keine Arbitrierung oder ähnliches stattfinden muss. Allerdings leidet die Flexibilität, da ein Master nicht mehr auf jeden, sondern nur noch auf einen einzigen Slave zugreifen kann. Für die meisten OFDM Systeme ist dies jedoch in der Regel ausreichend, da durch die Dual Port AHB Speicher (siehe Kapitel 4.4.2) ein Weiterreichen der Daten von Beschleuniger zu Beschleuniger möglich ist, vergleiche auch Abbildung 4.12. Eine Beteiligung des CS an der Signalverarbeitung ist aufgrund der fehlenden Unterstützung für die CS zu SPS Brücke in diesem Modus allerdings nicht mehr möglich.

In Tabelle 4.4 sind die Syntheseergebnisse des Interconnects im Matrix Modus (M-M) als auch im P2P-Modus (P2P-M) abgebildet. Man kann klar erkennen, dass im Matrix-Modus bei einer Vergrößerung der verfügbaren Schnittstellen der Ressourcenverbrauch stärker als linear ansteigt und die maximale Taktfrequenz abnimmt. Zu bedenken bleibt, dass gerade durch die Möglichkeit der Verwendung von Streams (siehe Kapitel 4.2.5) wenige Designvarianten mit besonders großen Matrizen benötigt werden. So dürfte in der Regel eine 16x16 Matrix bereits ausreichend sein.

Im P2P-Modus steigt der ohnehin geringe Ressourcenverbrauch linear, die Taktfrequenz ist von der Anzahl der Schnittstellen erwartungsgemäß unabhängig. Allerdings ist gerade hier der maximale Takt, welcher von den Synthesewerkzeugen angezeigt wird, wenig aussagekräftig, da in diesem Modus nur wenig Logik vorhanden ist und je nach Design die Verdrahtung weite Distanzen überbrücken muss. Die maximale Datenrate besteht aus den akkumulierten maximalen Daten-

Tabelle 4.4.: Virtex-6 Syntheseergebnisse des Multilayer Connectors im Matrix Modus (M-M) sowie Punkt-zu-Punkt Modus (P2P-M) für versch. Konfigurationen. Es werden weder BRAM noch DSP48E1 Primitive benötigt.

Konfiguration (#Master x #Slave)	Ressourcentyp		Maximale Taktfrequenz	Akkumulierte max. Datenrate
	LUTs	Flip Flops		
16 x 16 (M-M)	18 425	5 671	224 MHz	213,6 Gb/s
16 x 32 (M-M)	34 516	6 949	214 MHz	204,1 Gb/s
32 x 16 (M-M)	38 478	7 999	213 MHz	203,1 Gb/s
32 x 32 (M-M)	77 041	14 787	202 MHz	385,3 Gb/s
32 x 64 (M-M)	143 696	17 812	178 MHz	339,5 Gb/s
64 x 32 (M-M)	159 884	30 557	191 MHz	364,3 Gb/s
64 x 64 (M-M)	288 504	39 002	149 MHz	568,4 Gb/s
16 x 16 (P2P-M)	32	47	1 497 MHz	1,39 Tb/s
32 x 32 (P2P-M)	64	95	1 497 MHz	2,79 Tb/s
64 x 64 (P2P-M)	128	191	1 497 MHz	5,58 Tb/s

raten aller möglichen Verbindungen, bei den durch die Synthese spezifizierten maximalen Systemtaktraten.

4.4.2. Dual Port AHB Speicher

Um einen Dual Port AHB Speicher mit 64-Bit Wortbreite aufzubauen, werden mindestens zwei BRAM benötigt. Ein Wort wird dabei in ein 32-Bit High-Wert und ein 32-Bit Low-Wert aufgeteilt, welche dann jeweils in einem separatem BRAM gespeichert werden. Die Gesamtanzahl der verwendeten BRAM hängt von der eingestellten Speichertiefe ab, welche sich wieder per VHDL-Generic zur Designzeit einstellen lässt.

Mit Hilfe der beiden nativen Schnittstellen der BRAMs besitzt ein solcher AHB Speicher zwei getrennte, unabhängige 64-Bit AHB Slave Schnittstellen. Dadurch belegt ein Speicher zwei Schnittstellen am Interconnect, wodurch zwei Master parallel auf den selben Speicherbereich zugreifen können. So ist es zum Beispiel möglich, dass ein Beschleuniger seine Ergebnisse noch in einen AHB Speicher schreibt, während gleichzeitig der in der Signalverarbeitungskette nachfolgende Beschleuniger diese Ergebnisse bereits als neue Eingangsdaten einliest.

Tabelle 4.5 zeigt den durch die Synthesewerkzeuge ermittelten Ressourcenverbrauch sowie die ermittelte maximale Taktfrequenz. Wie zu erwarten steigt der BRAM Verbrauch mit der Speichertiefe linear an. Die Verwendung von restlicher Logik zur Steuerung und Organisation der BRAMs sowie zur Implementierung

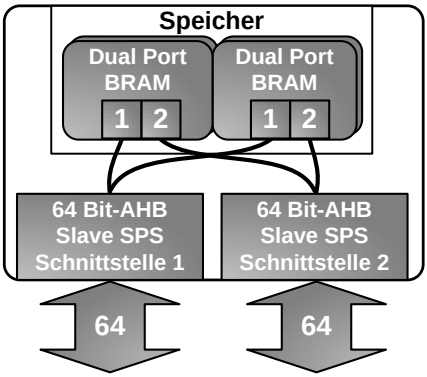


Abbildung 4.13.: Die Dual Port AHB Speicher sind durch mindestens 2 BRAM Blöcke realisiert, wobei der BRAM-Verbrauch letztendlich von der eingestellten Speichertiefe abhängt.

der AHB Slave Schnittstellen steigt erwartungsgemäß nur wenig mit zunehmender Speichertiefe. Dank der für Speicherzwecke optimierten BRAM Blöcke erreichen die AHB BRAM Speicher besonders hohe Taktraten. Wie immer muss aber auch hier bedacht werden, dass die Angaben der Synthese keine Aussage über die Verdrahtung machen. Durch die Verteilung der BRAM Blöcke über den gesamten Chip kommen gerade beim Zusammenschalten vieler solcher Komponenten lange Pfade zustande, die dann oftmals als längster kritischer Pfad die maximale Taktfrequenz bestimmen. Das Blockschaltbild eines Speichers ist in Abbildung 4.13 zu sehen.

Tabelle 4.5.: Virtex-6 Syntheseergebnisse der AHB Speichermodule für verschiedene Speichertiefen. Sie benötigen keine DSP48E1.

Speichertiefe in 64-Bit Wörtern	Speicherplatz in KB	Ressourcentyp			Maximale Taktfrequenz
		LUTs	Flip Flops	BRAMs	
1024	8	27	24	2	618 MHz
2048	16	29	26	4	612 MHz
4096	32	31	28	8	602 MHz
8192	64	33	30	16	583 MHz
16384	128	35	32	32	560 MHz
32768	256	37	34	64	545 MHz

4.4.3. PRBS-Generator und PRBS-Checker

Um die entwickelte Architektur zu evaluieren und Aussagen über die Übertragungsqualität eines entsprechenden Übertragungssystems vorzunehmen, werden geeignete Sendedaten in Form eines binären Datenstroms benötigt. Üblicherweise werden hierzu pseudozufällige Bitfolgen verwendet, *Pseudorandom Binary Sequence* (PRBS) genannt. Sie besitzen annähernd die Charakteristik von weißem Rauschen und sind streng deterministisch, was eine Überprüfung der Empfangsdaten am Empfänger ermöglicht.

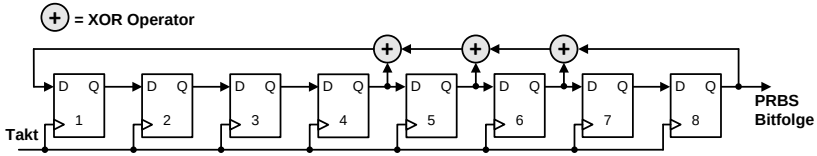


Abbildung 4.14.: Rückgekoppeltes Schieberegister 8ten Grades zur Erzeugung pseudo-zufälliger Bitfolgen.

Ein weiterer Vorteil ist, dass PRBS Daten sich technisch recht einfach erzeugen lassen. Es wird lediglich ein linear rückgekoppeltes Schieberegister benötigt, die Rückkopplungen verwenden die Antivalenz als logische Funktion. Die Position der Rückkopplungen sowie der Grad des Polynoms hat Einfluss auf die maximale Periodenlänge der erzeugten sich wiederholenden Bitfolge. Ein entsprechendes Schieberegister achten Grades ist in Abbildung 4.14 aufgezeigt.

Eine Standard-Realisierung wie in Abbildung 4.14 dargestellt, erzeugt jeden Taktzyklus ein weiteres Bit des PRBS Bitstroms. Bedenkt man, dass höherwertige Modulationsformate mehrere Bit pro Takt modulieren und jeder Beschleuniger zwei Samples pro Takt verarbeiten soll, dann wird schnell klar, dass man mehrere Bits pro Takt erzeugen muss. Dies gilt erst recht wenn mehrere SPS-Streams im System realisiert werden sollen.

Tabelle 4.6.: Virtex-6 Syntheseergebnisse für den PRBS Generator Beschleuniger und den PRBS Checker Beschleuniger. Sie benötigen beide weder BRAM noch DSP48E1 Primitive

	Ressourcentyp		Maximale
Beschleuniger	**LUTs**	**Flip Flops**	**Taktfrequenz**
PRBS Generator	796	560	297 MHz
PRBS Checker	877	540	235 MHz

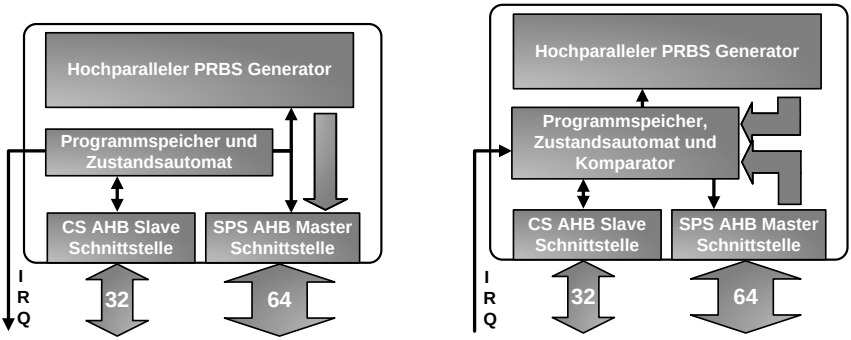


Abbildung 4.15.: Trotz des ähnlichen Aufbaus unterscheidet sich der PRBS Generator Beschleuniger stark in der Steuerlogik vom PRBS Checker Beschleuniger.

Ein PRBS Generator, welcher pro Taktzyklus n-Bits ausgibt, lässt sich in einem FPGA erzeugen, indem ausgehend vom aktuellen Schieberegistertakt nicht nur der nächsten Zustand, sondern mittels kombinatorischer Logik gleich die nächsten n-Zustände berechnet werden. Das Schieberegister erhält am Ende des Taktes den Wert der Berechnung für Stufe n. Um Sendedaten für Experimente zur Verfügung zu stellen sowie um die Empfangsdaten auf eine korrekte Übertragung zu überprüfen, wurden zwei spezielle Beschleuniger entwickelt, der *PRBS Generator Beschleuniger* sowie der *PRBS Checker Beschleuniger*, siehe Abbildung 4.15.

Die Schieberegister der Beschleuniger lassen sich mit beliebigen *Seeds* (Initialwerte) ausstatten und sind 32-fach parallel ausgeführt. So werden pro Takt 32 Bit des PRBS Bitstromes erzeugt. Der PRBS Generator Beschleuniger benötigt folglich zwei Takte, um 64 Bit PRBS Daten zu erzeugen, welche er anschließend über die AHB Master Schnittstelle über den Bus in eine Speicheradresse schreiben kann. Über sein Programm lässt sich bestimmen, wie viele 64-Bit Wörter PRBS Daten er generieren und wohin er sie schreiben soll. Am Ende der Iteration können Interrupts erzeugt werden, entweder auf dem SPS IRQ Bus, um beispielsweise dem nachfolgenden Beschleuniger die Verfügbarkeit neuer Daten zu signalisieren, oder auf dem CS Bus, um den Prozessor des CS über die Erfüllung der erteilten Aufgabe zu informieren.

Der PRBS Checker Beschleuniger, rechts im Bild, ist ähnlich aufgebaut wie der PRBS Generator Beschleuniger. Allerdings schreibt er die erzeugten 64-Bit PRBS Wörter nicht in einen Speicher, sondern vergleicht sie mit den Daten, welche er über seine AHB Master Schnittstelle von einer Speicheradresse gelesen hat. Er merkt sich dabei in getrennten 64-Bit breiten Zählern, wie viele Bits übereinstim-

men und wie viele Bits unterschiedlich sind. Bevor einer der beiden Zähler überläuft, werden beide Zählerstände durch zwei dividiert und bei Bedarf ein Interrupt erzeugt. In Tabelle 4.6 sind die Syntheseergebnisse für den Ressourcenverbrauch und die maximale Taktrate für beide Beschleuniger angegeben.

4.4.4. Modulations- und Demodulationsbeschleuniger

Das Zuordnen der zu übertragenden Sendebits zu entsprechenden Punkten im Konstellationsdiagramm ist die Aufgabe des Modulationsbeschleunigers, das Gegenstück für den Empfänger bildet der Demodulationsbeschleuniger. Das Blockdiagramm des Modulationsbeschleuniger ist in Abbildung 4.16 dargestellt.

Die binären Daten, welche der Beschleuniger mit seiner ersten AHB Master Schnittstelle aus einem Speicher ausliest, werden zunächst in einem 128 Bit breiten Puffer zwischengespeichert. Der Pufferspeicher ermöglicht es, dass immer genügend Bit vorrätig sind, egal welche Modulationsarten im nächsten Takt für die zwei zu verarbeitenden Samples umgesetzt werden. Sinkt der Füllstand des Pufferspeichers unter eine definierte Mindestanzahl an Bit, werden neue Quelldaten eingelesen und der Puffer aufgefüllt. Dies geschieht so lange, bis im Befehlsspeicher des Hardwarebeschleunigers keine Modulationen mehr vorgesehen sind.

Der Modulationsbeschleuniger hat neben den Standardregistern zur Programmierung der Ablaufsteuerung und neben dem Programmspeicher zwei weitere Speicherbereiche. Der *OFDM Konfigurationsspeicher* stellt ein Teil des Programmspeichers dar und enthält die Information, welche OFDM-Subträger wie moduliert werden müssen. Derzeit unterstützt der Beschleuniger unmodulierte Subträger, Pilotträger sowie die Modulationsformate QPSK, 16-QAM und 64-QAM. Im zweiten zusätzlichen Speicherbereich, dem Pilotspeicher, wird eine individuelle Anzahl an Pilotwerten abgelegt. Mit Hilfe dieses Speichers können beliebige Muster für die Pilottöne realisiert werden.

Die Modulation selbst findet in den zwei Mapping Modulen statt, welche jeweils, je nach aktueller Modulationsart, zwischen 0 (unmoduliert oder Pilotton) und 6 Bit (64-QAM) pro Takt aus dem Pufferspeicher lesen. Ist der aktuelle Träger unmoduliert, wird der Wert Null ausgegeben, ist es ein modulierter Träger, wird in Modulationstabellen der korrekte Ausgabewert zu den Eingangsdaten ermittelt. Handelt es sich beim aktuellen Subträger um einen Pilotenträger, wird der nächste Pilotwert aus dem Pilotspeicher ausgegeben. Wurden alle Werte des Pilotspeichers verwendet, springt die Leseadresse wieder auf Adresse Null und die Werte wiederholen sich.

Der Demodulationsbeschleuniger ist im Aufbau dem Modulationsbeschleuniger nachempfunden, allerdings ist er nicht ganz so komplex wie sein Gegenstück. Jedes der zwei *Demodulation Modules* erhält ein komplexes Sample pro Takt von

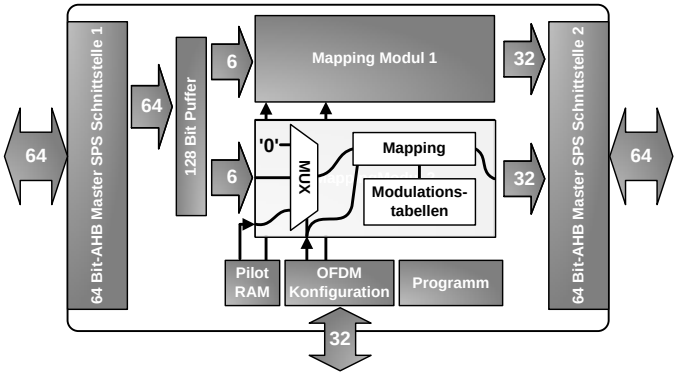


Abbildung 4.16.: Blockschaltbild des Modulationsbeschleunigers. Neben dem üblichen Konfigurations- bzw. Programmspeichern besitzt er zusätzliche Speicher welche die Modulationsarten der Subträger sowie die Werte der Piloten bestimmen.

der ersten Master Schnittstelle und liest die Modulationsart für dieses Sample aus dem entsprechenden Konfigurationsspeicher. Für das aktuelle Sample wird dann jeweils bestimmt, in welchem Gültigkeitsbereich der aktuellen Modulationsart sich dieses Sample im Konstellationsdiagramm befindet.

Die Gültigkeitsbereiche einer Modulationsart sind zu Quadraten festgelegt. Die Zugehörigkeit eines Samples zu einem Bereich kann bestimmt werden, indem der Realteil und der Imaginärteil des Samples getrennt mit den existierenden Grenzen der Gültigkeitsbereiche für die jeweilige Modulationart verglichen werden. Pilottöne und unmodulierte Träger werden ignoriert. Abbildung 4.17 zeigt den Aufbau des Demodulationsbeschleunigers mit einer Darstellung der Gültigkeitsbereiche für eine 16-QAM Modulation.

Der Ressourcenverbrauch der Beschleuniger ist in Tabelle 4.7 angegeben. Die gelisteten BRAMs werden zum Speichern der Konfigurationsdaten und der Pilotenwerte (Pilot RAM) verwendet. Die maximale Tiefe dieser Speicher und damit die Anzahl der verwendeten BRAMs wird zur Designzeit durch VHDL-Generics festgelegt.

4.4.5. (i)FFT-Beschleuniger

Der (i)FFT-Beschleuniger spielt eine besondere Rolle im System, da zum einen die Subträgeranzahl neben der Datenrate einen der maßgebenden OFDM-Parameter

Tabelle 4.7.: Virtex-6 Syntheseergebnisse für den Modulations- sowie den Demodulationsbeschleuniger. Beide Beschleuniger verwenden keine DSP48E1 Primitive.

Beschleuniger	Ressourcentyp			Maximale
	LUTs	Flip Flops	BRAMs	Taktfrequenz
Modulationsbeschleuniger	2282	1281	2	235 MHz
Demodulationsbeschleuniger	1253	869	1	286 MHz

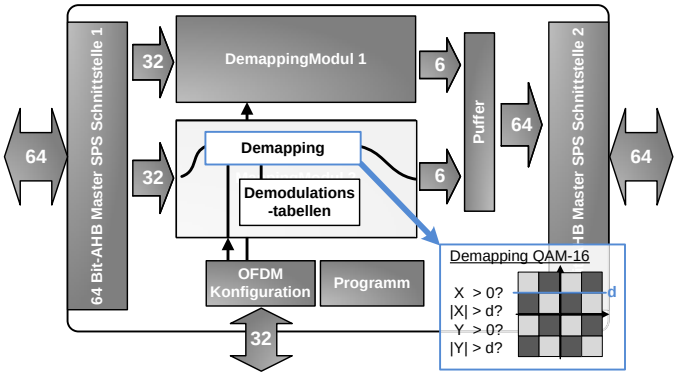


Abbildung 4.17.: Der Demodulationsbeschleuniger nutzt Tabellen mit den Werten der Bereichsgrenzen um durch Komparatoren den zugehörigen Gültigkeitsbereich eines Samples zu bestimmen.

darstellt, zum anderen, da der Algorithmus, je nach Subträgeranzahl, oftmals die Komplexität und den Ressourcenverbrauch eines ganzen OFDM-Systems dominieren kann. Die grundsätzlichen Anforderungen an einen solchen Beschleuniger wurden in [Wit12] erarbeitet. Beim hier vorgestellten Beschleuniger handelt es sich um eine angepasste und stark erweiterte Version des Beschleunigers dieser Arbeit.

4.4.5.1. Aufbau und Konzept

Der (i)FFT-Beschleuniger soll, wie alle Beschleuniger des SPS, möglichst flexibel sein und einen Durchsatz von zwei Samples pro Takt bieten. Für einen FFT-Hardwarebeschleuniger heißt Flexibilität in erster Linie, möglichst viele verschiedene Anzahlen an Subträgern realisieren zu können.

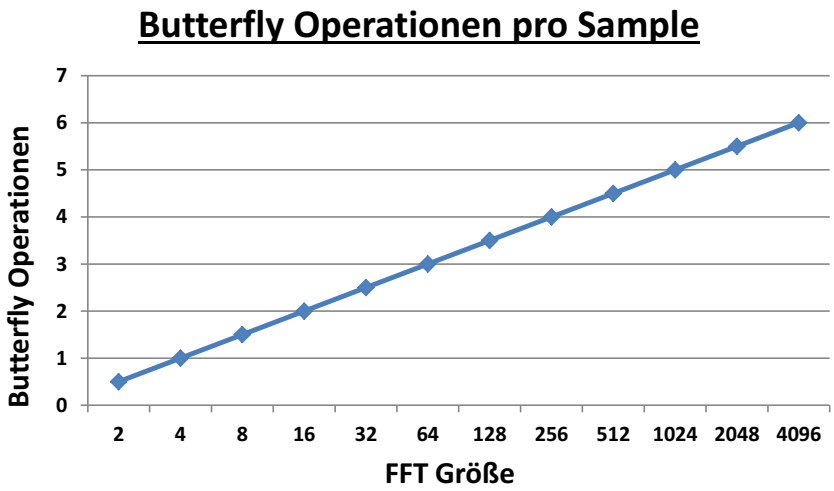


Abbildung 4.18.: Die möglichen Transformationsgrößen einer Radix-2 FFT sind durch Zweierpotenzen gegeben. Die Anzahl der benötigten Butterflies pro Sample nimmt daher logarithmisch zu.

Der hierzu entwickelte (i)FFT-Beschleuniger verwendet eine Radix-2 Implementierung des FFT-Algorithmus. Der Vorteil liegt hier in der höheren Flexibilität im Vergleich zu höherwertigeren Radix-Implementierungen, da in der Regel nur Transformationsgrößen als Potenz der Radix möglich sind. Mit der Größe der Transformation steigt natürlich auch die Anzahl der benötigten Butterfly Operationen an. Die Anzahl an Radix-2 Butterfly Operationen ist hierbei durch Formel 4.2 gegeben.

$$\begin{aligned} \text{Anzahl Radix-2 für FFT der Größe X} &= \text{X}/2 \cdot \text{Anzahl Stufen} \\ &= \text{X}/2 \cdot ld(X) \end{aligned} \tag{4.2}$$

Die Anzahl an benötigten Operationen steigt jedoch nicht linear mit den Samples, sondern logarithmisch an. Dieser Anstieg ist gut in Abbildung 4.18 zu sehen, wenn man bedenkt, dass die Achse der FFT-Größe exponentiell dargestellt ist.

Wie in Kapitel 2.1.3 veranschaulicht, besteht eine FFT aus mehreren Stufen mit meist mehreren Butterfly Operationen pro Stufe. Ein Radix-2 Butterfly benötigt für eine Operation zwei komplexe Samples und erzeugt auch als Ergebnis zwei

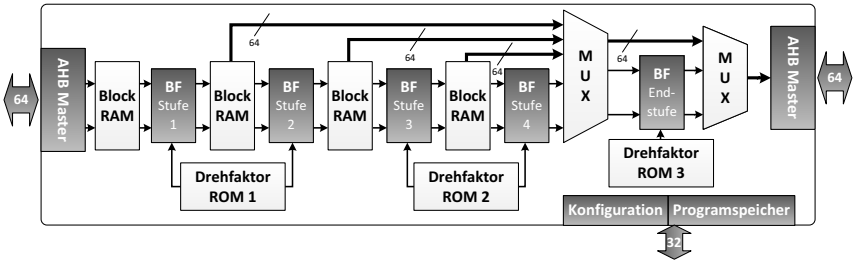


Abbildung 4.19.: Der (i)FFT-Beschleuniger besitzt 4+1 Stufen und kann mit anderen Beschleunigern dieses Typs gemeinsam an einer Transformation rechnen.

komplexe Samples. Da auch der angestrebte Durchsatz bei zwei Samples pro Taktzyklus liegt, wird pro Stufe genau eine Butterfly Einheit benötigt. Die resultierenden zwei Samples einer Stufe werden dann einer eventuell nachfolgenden Stufe übergeben.

Dabei sollten innerhalb eines Beschleunigers nicht zu viele Butterfly-Einheiten realisiert sein, da bei kleineren (i)FFT Größen ansonsten überflüssige Stufen nicht genutzt werden. Um dennoch flexible Transformationsgrößen realisieren zu können bietet der entworfene Beschleuniger die Möglichkeit, mit anderen (i)FFT-Beschleunigern gemeinsam eine Transformation zu berechnen. Der Hardwarebeschleuniger realisiert dabei 4+1 Stufen, wobei die letzte Stufe auch nur für die letzte Stufe einer Transformation verwendet werden kann. In Abbildung 4.19 ist das Blockschaltbild des Beschleunigers dargestellt.

Die ersten vier Butterflystufen eines Beschleunigers werden mit vorangehenden BRAMs gepuffert. Dies ist notwendig, um sicher zu stellen, dass die benötigten Samples für die nächste Operation zur Verfügung stehen. So reicht zwar ein Buszugriff aus um zwei Samples pro Takt zu erhalten, allerdings handelt es sich hierbei immer um aufeinanderfolgende Sample-Paare. Entsprechend müssen die Butterfly-Einheiten teilweise ihren Start verzögern, bis das gewünschte Sample-Paar im Speicher vorhanden ist. Dank dem Pipelining-Prinzip erreicht der Beschleuniger dennoch einen Durchsatz von zwei Samples pro Takt, sobald seine Pipeline gefüllt ist.

Die Drehfaktoren der Butterfly-Stufen werden in Read-Only Speicher (ROM) gelagert, deren Inhalt bereits zur Designzeit festgelegt wird. Dabei hängt die benötigte Speichertiefe der ROMs von der entsprechenden Stufe ab. Zudem können sich auf Grund der zwei unabhängigen Ports der BRAM immer zwei der Standard Stufen einen Drehfaktor-ROM teilen. Eine Besonderheit ist durch die letzte

Stufe des Prozessors gegeben. Sie ist nur für die letzte Stufe einer (i)FFT Berechnung vorgesehen, braucht daher zwar den größten Speicher für die Drehfaktoren, kommt aber ohne Puffer RAM aus, da die Ergebnisse der vorletzten Stufe immer in der richtigen Reihenfolge geliefert werden.

Ein einzelner (i)FFT-Beschleuniger ist somit in der Lage eine 32-Punkte FFT mit einem Durchsatz von zwei Samples pro Takt zu berechnen. Um auch weniger Stufen zu realisieren, können alle Stufen außer der ersten, durch entsprechende Multiplexer umgangen werden. Sollen Fourier Transformationen mit mehr als 32 Stützstellen berechnet werden, kann dies mittels zwei Alternativen geschehen.

Eine Möglichkeit besteht darin, die Daten mehrmals mit einem einzigen Hardwarebeschleuniger zu bearbeiten. Selbstverständlich führt dies jedoch zu einer Reduktion des Durchsatzes bezogen auf die gesamte Transformation, was in der Regel unerwünscht ist.

Befinden sich mehrere (i)FFT-Beschleuniger in dem System ist es möglich, mit mehreren Beschleunigern gemeinsam an einer Transformation zu rechnen. Für eine 128-Punkte Transformation werden dann beispielsweise zuerst die ersten vier Stufen durch einen der Beschleuniger berechnet, die letzte Stufe umgangen und das Ergebnis dann in einem AHB Speicher des Systems geschrieben. Ein zweiter Beschleuniger holt sich diese Daten anschließend und berechnet die fünfte und sechste Stufe in seinen lokal ersten zwei Butterfly-Stufen und gibt die Ergebnisse dann zur letzten Stufe, um in ihr die letzte Stufe der Transformation zu berechnen, vergleiche Abbildung 4.20.

Welche Stufen einer Transformation ein Beschleuniger berechnet wird dabei durch das Programm vorgegeben. So kann für die Berechnung wieder ein Datendurchsatz von zwei Samples pro Takt, bezogen auf die gesamte Transformation, erreicht werden.

Ob es sich bei der Berechnung um eine normale oder um eine Inverse Fourier Transformation handelt und somit die Real- und Imaginärteile der Eingangs- als auch der Ausgangsdaten vertauscht werden müssen, wird durch das Programm des Beschleunigers gesteuert, vgl. Kapitel 2.1.3. Vor jeder Butterflystufe kann, ebenfalls durch das Programm gesteuert, eine optionale Division durch Zwei oder Vier erfolgen, vergleiche Kapitel 4.4.5.2. Dies kann zwar verwendet werden, um das benötigte Skalieren des Ergebnisses einer inversen FFT zu erreichen (siehe Kapitel 2.1.3), allerdings wird dies bei Kommunikationssystemen nicht unbedingt benötigt, da der absolute Wert bei der Übertragung ohnehin verloren geht.

Eine Entscheidung, welche zur Designzeit getroffen werden muss, ist die Variante des Hardwarebeschleunigers, beziehungsweise die Art und Weise, wie der Beschleuniger die komplexen Multiplikation innerhalb einer Butterflystufe realisieren soll. Standardmäßig wird diese wie in Gleichung 4.3 beschrieben, mit vier Multiplikationen, einer reellen Additionen und einer reellen Subtraktion durch-

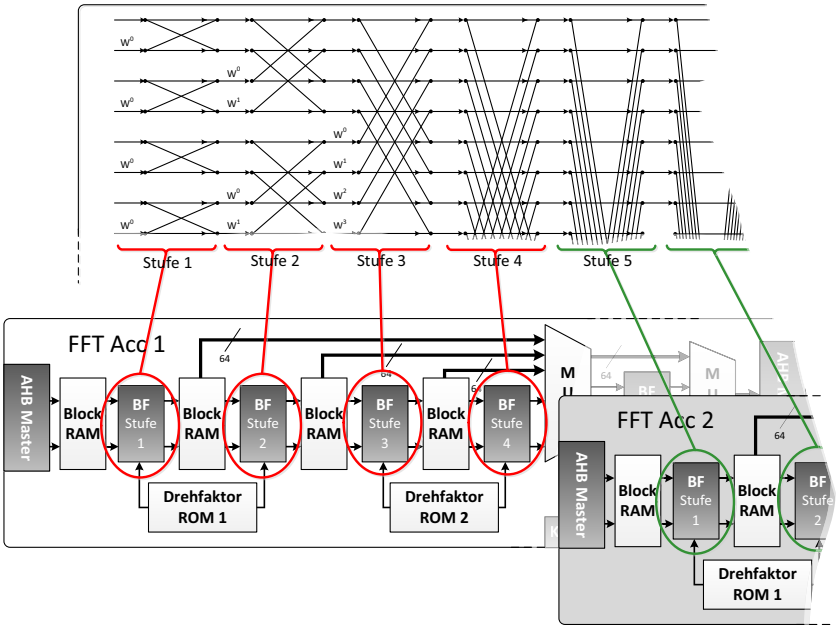


Abbildung 4.20.: Berechnung einer FFT durch mehrere (i)FFT-Beschleuniger: Der erste (i)FFT-Beschleuniger in der Verarbeitungskette berechnet die ersten vier Stufen der Transformation (rot). Die weiteren Berechnungsstufen werden von nachfolgenden (i)FFT-Beschleunigern übernommen (grün).

geführt. Abbildung 4.21 zeigt das Blockschaltbild der Butterfly Realisierung nach diesem Konzept. Dabei sind die Operationen, welche in einem DSP abgebildet werden können blau markiert. Es kommen entsprechend vier DSPs zum Einsatz.

$$\begin{aligned} \underline{z} = \underline{x} \cdot \underline{y} &= (x_{Re} + j \cdot x_{Im}) \cdot (y_{Re} + j \cdot y_{Im}) \\ &= \underbrace{x_{Re}y_{Re} - x_{Im}y_{Im}}_{z_{Re}} + j \cdot \underbrace{(x_{Re}y_{Im} - x_{Im}y_{Re})}_{z_{Im}} \end{aligned} \tag{4.3}$$

Es ist zwar möglich, Gleichung 4.3 so umzuformen, dass diese auf drei Multiplikationen reduziert wird, allerdings werden dann mehr Additionen und Subtraktionen benötigt. Die mathematische Herleitung hierzu findet sich in den Glei-

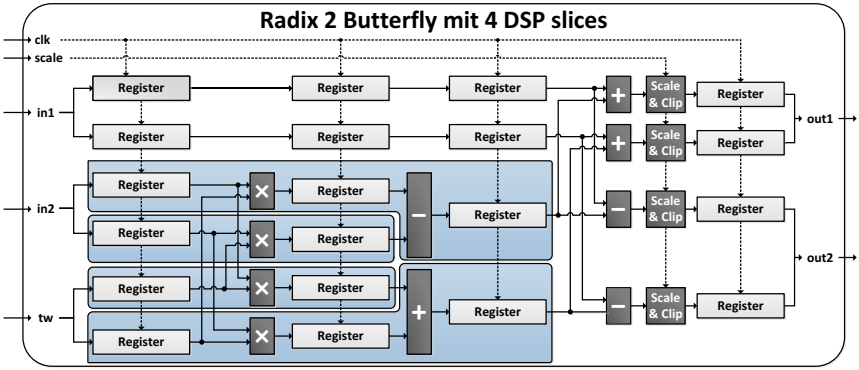


Abbildung 4.21.: Die Butterfly-Realisierungen der 4-DSP Version des (i)FFT-Beschleuniger nutzen das Pipelining Prinzip und bieten die Möglichkeiten der Umskalierung und Begrenzung von Ergebnissen um Über- und Unterläufe zu behandeln oder zu vermeiden. Die blauen Module kennzeichnen die verwendeten vier DSPs.

chungen 4.4 und 4.5, wobei das jeweilige Ergebnis aus Gleichung 4.3 durch eine Erweiterung auf die neue Form gebracht wird.

$$\begin{aligned} z_{Re} &= x_{Re}y_{Re} - x_{Im}y_{Im} \\ &= x_{Re}y_{Re} - x_{Im}y_{Im} + x_{Im}y_{Re} - x_{Im}y_{Re} \\ &= \underbrace{(y_{Re} - y_{Im}) \cdot x_{Im}}_{P} + \underbrace{(x_{Re} - x_{Im}) \cdot y_{Re}}_{Q} \end{aligned} \tag{4.4}$$

$$\begin{aligned} z_{Im} &= x_{Re}y_{Im} + x_{Im}y_{Re} \\ &= x_{Re}y_{Im} + x_{Im}y_{Re} + x_{Re}y_{Re} - x_{Re}y_{Re} \\ &= \underbrace{(y_{Re} + y_{Im}) \cdot x_{Re}}_{R} - \underbrace{(x_{Re} - x_{Im}) \cdot y_{Re}}_{Q} \end{aligned} \tag{4.5}$$

Der Q-Teil muss nur einmal berechnet werden. So kommt man hier mit drei Multiplikationen aus. Durch Verwendung der Pre-Adder und Post-Adder der DSP-slices kann die komplexe Multiplikation komplett in drei DSPs realisiert werden, vergleiche Abbildung 4.22. Allerdings ist zu bedenken, dass diese Variante nicht

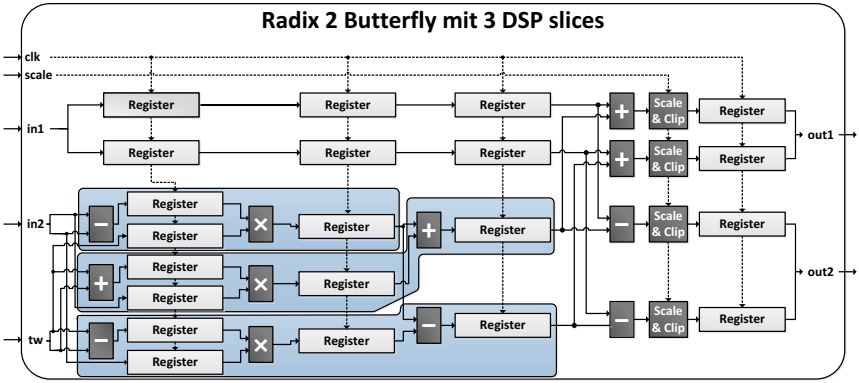


Abbildung 4.22.: Die zweite Version der Butterfly-Beschreibung lässt sich durch 3 DSPs realisieren. Allerdings kann sie nicht so schnell getaktet werden. Auch hier kennzeichnen die blauen Module die verwendeten DSPs.

nur mehr Additionen/Subtraktionen benötigt, sondern auch eine höhere Latenz hat. Um die gleiche Latenz der ersten Version zu erreichen, fehlt demnach die Eingangs-Registerstufe zur Optimierung der Verarbeitungsgeschwindigkeit. Gerade wenn man *Retiming* beziehungsweise *Register Balancing* verwendet, wenn also die Register innerhalb der Logik verschoben werden dürfen, um die maximale Taktrate zu erhöhen, kann es vorkommen, dass die Register und evtl. auch die Additionen/Subtraktionen nicht mehr in den DSPs verwirklicht werden, wie das in Abbildung 4.21 und 4.22 dargestellt ist. Hierdurch lässt sich auch der unterschiedliche Ressourcenverbrauch der verschiedenen Versionen sowie die unterschiedliche maximale Taktfrequenz, vergleiche Tabelle 4.8 erklären.

4.4.5.2. Rechengenauigkeit des (i)FFT-Beschleunigers

Neben des mit der Anzahl an Stützstellen steigenden Bedarfs an Butterfly-Operationen steigt auch die Anzahl an Bits die benötigt wird, um keine Rechenfehler bei den durchzuführenden Fixkomma-Operationen zu erhalten. Während man die Auflösung zwar innerhalb des (i)FFT theoretisch beliebig erhöhen kann, müssen die Ergebnisse spätestens beim Zurückschreiben in den Speicher wieder auf eine Auflösung von 16-Bit reduziert werden. Deshalb würde ein starkes Erhöhen der Bitbreite innerhalb eines (i)FFT-Beschleunigers die Genauigkeit der Berechnung nur leicht erhöhen, der Ressourcenverbrauch würde aber drastisch ansteigen, da

Tabelle 4.8.: Syntheseergebnisse des (i)FFT Hardwarebeschleuniger für die Virtex-6 Architektur.

Maximale (i)FFT Größe	BF-Version	Ressourcentyp				Maximale Taktfrequenz
		LUTs	Flip Flops	BRAMs	DSP48E1	
32	4-DSP	5652	3923	14	20	285 MHz
512	4-DSP	6274	4223	15	20	285 MHz
8192	4-DSP	6746	4369	32	20	285 MHz
65536	4-DSP	7116	4489	288	20	285 MHz
32	3-DSP	6889	5237	14	15	197 MHz
512	3-DSP	7444	5418	15	15	197 MHz
8192	3-DSP	7952	5627	32	15	197 MHz
65536	3-DSP	8545	5778	288	15	166 MHz

beispielsweise für das Speichern der Zwischenergebnisse breitere Speicher benötigt würden.

Da die komplexe Multiplikation mit den Drehfaktoren die Amplitude des komplexen Raumzeigers nicht verändert, spielen für die Bitgrößen die Additionen und Subtraktionen die größte Rolle. Als Kompromiss zwischen Ressourcenverbrauch und Genauigkeit reduziert der Beschleuniger daher die Auflösung des Ergebnisses nach jeder Butterflystufe auf 18-Bit. Dies erhöht die benötigte Anzahl der BRAMs nicht, da die vier zusätzlichen Bits pro komplexer Zahl in den ECC-Bits der 32-Bit breiten BRAMs gespeichert werden können. Nach der letzten Stufe wird das Ergebnis wieder auf 16-Bit reduziert, um es im AHB Speicher abzulegen.

Innerhalb einer Butterflystufe wird während einer Berechnung durch Erhöhen der Auflösung das Ergebnis vor Überläufen und Rundungsfehlern geschützt. Dies ist ohne erheblichen Mehraufwand an Ressourcen möglich, da die DSP Einheiten im FPGA ohnehin mit einer entsprechenden Auflösung realisiert sind. Dabei wird das Komma automatisch um zwei Stellen verschoben, was einer Division durch vier entspricht. Am Ende der Berechnung, wenn das Ergebnis auf 18 Bit reduziert werden muss, sind die gängigen Methoden implementiert, einen Überlauf oder einen Unterlauf zu behandeln beziehungsweise zu vermeiden, ohne den Wertebereich zu erhöhen, vergleiche [82].

Zum einen werden die Ergebnisse in der letzten Pipelinestufe (Scale & Clip) eines Butterfly zum Entfernen der unteren Bits gerundet, was im Vergleich zum einfachen Abschneiden der Bits die Genauigkeit erhöht. Zum anderen bietet jede Butterfly-Einheit eines (i)FFT-Beschleunigers individuell die Möglichkeit, das Ergebnis um den Faktor 2 oder 4 zu skalieren bzw. die automatische Division zu verhindern. Dies kann zwar die Genauigkeit erhöhen, allerdings wird hierdurch ein Überlaufen beziehungsweise ein Unterlaufen des Ergebnisses ermöglicht. Die Be-

grenzung des Wertebereichs der Ergebnisse (Clipping) sorgt aber für eine Reduzierung dieser Fehler. In welchen Stufen die Division durchgeführt werden sollte lässt sich nicht generell festlegen. Dies hängt von der Größe der Transformation sowie von den Eingangsdaten ab.

4.4.6. Equalization-Beschleuniger

Der Equalization-Beschleuniger wird nur in OFDM-Empfängern benötigt und hat den Zweck, die Konstellationspunkte zu entzerren und für eine Demodulation korrekt zu skalieren. Der Hardwarebeschleuniger ist zweistufig aufgebaut, wobei die erste Stufe eine präambelbasierte Korrektur durchführt, die zweite Stufe ihre Korrekturfaktoren anhand von Pilottönen berechnet und daher auch durchgehend nachführt.

Tabelle 4.9.: Syntheseergebnisse des Equalization-Beschleuniger für Virtex-6.

	Ressourcentyp				Maximale
Modul	**LUTs**	**Flip Flops**	**BRAMs**	**DSP48E1**	**Taktfrequenz**
Equalization-Beschl.	23714	15512	11	60	129 MHz
Komplexe Division	5984	4098	0	6	229 MHz

Zu Beginn eines jeden Blocks teilt die erste Stufe die aus dem Präambelspeicher bekannte, gesendete Präambel durch die empfangene Präambel, wodurch sich die statischen Korrekturfaktoren ergeben. Dabei kommt eine komplexe Division mit Hilfe eines Hardwaremoduls zum Einsatz, welche eine Vielzahl an Pipelinestufen nutzt, um bei gefüllter Pipeline mit jeden Takt ein Ergebnis zu liefern. Dieses Modul wird im folgenden *Divider* genannt.

Ein grobes Blockschaltbild eines solchen Dividers ist in Abbildung 4.23 dargestellt. Zuerst werden führende Nullen im Zähler als auch im Nenner durch Barrelshifter entfernt. Die Differenz der führenden Nullstellen muss in die Berechnung des Ergebnisses mit berücksichtigt werden, dies wird durch entsprechende Zähler erreicht. Anschließend wird in einer Kette von Subtraktionsmodulen der Nenner vom Zähler abgezogen, falls der Zähler größer oder zumindest gleich groß ist. Falls nicht, wird der Zähler für die nächste Stufe verdoppelt. Eine erfolgte Subtraktion oder eine Verdoppelung (Shift-Operation) fließt wieder in den Ergebniszähler mit einer Eins oder einer Null an der entsprechenden Stelle ein.

Im Detail befindet sich noch zusätzliche Logik in einem solchen Divider, um auch korrekt mit negativen Zahlen etc. umzugehen. Mit Hilfe solcher Divider wird dann entsprechend der Gleichung 4.6, mittels 6 Multiplikationen, 2 Divisionen, 2 Additionen und einer Subtraktion, ein komplexer Divider aufgebaut. Die Latenz

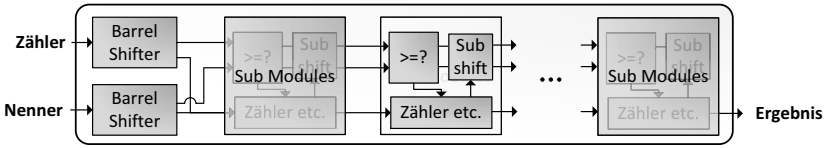


Abbildung 4.23.: Der Divider ist mittels Pipelining-Technik auf einen hohen Datendurchsatz und eine hohe Taktfrequenz optimiert.

eines komplexen Dividers beträgt dabei 40 Taktzyklen, sollte die EQ-Präambel wirklich kleiner sein als 40 Samples, dann muss durch entsprechende Maßnahmen, wie beispielsweise dem Aufteilen der EQ-Präambel auf mehrere Symbole, der Kanalschätzung genügend Zeit geschaffen werden.

$$\frac{a+jb}{c+jd} = \frac{ac+bd}{c^2+d^2} + j\frac{bc-ad}{c^2+d^2} \tag{4.6}$$

Ein Aufteilen der Präambel ist sinnvoll, um in den EQ-Präambel Symbolen nicht alle Träger modulieren zu müssen. Dies ermöglicht es, die Energie in diesen Symbolen niedrig zu halten. Ein Clipping beim Senden oder Empfangen dieser Symbole sollte unbedingt vermieden werden.

Sind alle EQ-Präambel Symbole eingelesen, folgen die Datensymbole, welche direkt mit den errechneten Korrekturfaktoren im ersten Equalizer entsprechend korrigiert werden, vergleiche das Blockschaltbild in Abbildung 4.24. Da diese Korrekturfaktoren sich innerhalb eines Blocks/Frames nicht mehr ändern, wird diese Korrekturstufe als statisch bezeichnet.

Anschließend werden die statisch korrigierten Symbolwerte aufgeteilt. Zum einen werden sie durch Ringspeicher, welcher aus BRAM-Primitiven erstellt ist, verzögert. Zum anderen ermittelt in der Zwischenzeit ein Zustandsautomat mit Hilfe der programmierten Pilottonkonfiguration die Korrekturwerte der Pilot-Subträger für die zweite Stufe, indem er die Referenz-Pilottöne durch die empfangenen Pilottöne teilt. Hierzu wird der selbe Hardware-Divider verwendet, welcher auch schon zur Berechnung der statischen Korrekturfaktoren dient.

Sobald die ersten zwei aktualisierten Korrekturwerte vorliegen wird eine lineare Extra- sowie Interpolation gestartet, um für die Datenträger ebenfalls neue Korrekturwerte zu erhalten. Sind genügend Werte vorhanden, werden diese in der zweiten, dynamischen Korrekturstufe auf das entsprechende OFDM Symbol angewandt. Das Prinzip der linearen Interpolation ist in Abbildung 4.25 dargestellt, weitere Informationen finden sich in [78] sowie [74].

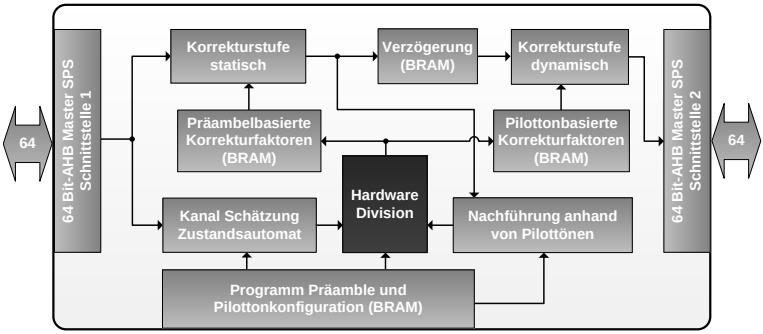


Abbildung 4.24.: Der Equalization-Beschleuniger bietet eine Präambel basierte Entzerrung, als auch eine über Pilottöne nachgeführte Entzerrung. Die ressourcenintensive Division kann dabei von beiden Teilen genutzt werden.

Bei der Korrekturstufe handelt es sich jeweils um eine komplexe Multiplikation der Eingangswerte mit den Korrekturwerten. Es findet somit in beiden Stufen eine Korrektur der Phase als auch der Amplitude statt. Ebenso ist anzumerken, dass auch dieser Beschleuniger einen Durchsatz von zwei Samples pro Takt erreicht und somit die meisten Komponenten wie die Division und die Korrekturstufen doppelt ausgeführt sind. Des Weiteren werden in der Standard Konfiguration die Bitbreiten in der zweiten Stufe auf 32-Bit erweitert, um Quantisierungsfehler zu minimieren.

Tabelle 4.9 zeigt die Syntheseergebnisse für den Equalization-Hardwarebeschleuniger für die Standard Konfiguration, sowie den Ressourcenverbrauch einer komplexen Hardware Division. Es ist leicht zu erkennen, dass die im Beschleuniger vorhandenen zwei Hardware Divisionen einen großen Teil der Ressourcen beanspruchen.

4.4.7. Data Output Converter und Synchronisationsbeschleuniger

Der Data Output Converter und der Synchronisationsbeschleuniger sind spezielle Hardwarebeschleuniger, die jeweils nur einmal im Sender beziehungsweise Empfänger benötigt werden. Sie führen nicht nur die letzten bzw. ersten Operationen an den Daten aus, sie sind außerdem direkt mit der analogen Schnittstelle verbunden wodurch sie lediglich eine einzige Master AHB Schnittstelle benötigen.

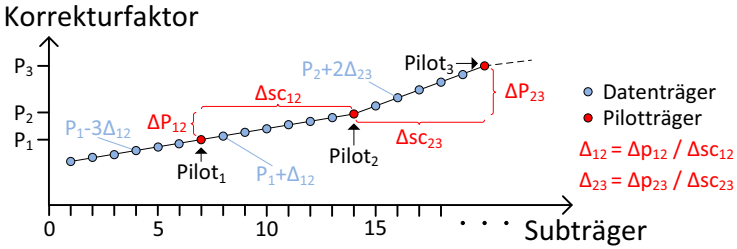


Abbildung 4.25.: Um Korrekturfaktoren für die Datenträger zu erhalten, wird zwischen den berechneten Korrekturfaktoren für die Pilottöne linear interpoliert und extrapoliert

Diese analogen Schnittstellen sind nicht Teil des Frameworks, da sie für jeden DAC, beziehungsweise ADC, neu entwickelt oder zumindest angepasst werden müssen. Der Grund hierfür ist, dass die Art und Weise, wie die Daten an den DAC geliefert werden müssen, beziehungsweise wie ein ADC dem System die Daten zur Verfügung stellt, sich von Wandler zu Wandler unterscheidet.

Es wird immer nur eine Instanz eines Data Output Converters und/oder eines Synchronisationsbeschleunigers benötigt. Diese muss allerdings die maximale Anzahl an zu unterstützenden SPS-Streams und somit an zu verarbeiteten Samples pro Taktzyklus unterstützen. Die Komplexität und der Ressourcenverbrauch wird also auch hier zur Designzeit über ein VHDL-Generic festgelegt.

4.4.7.1. Data Output Converter

Der Data Output Converter ist der letzte Beschleuniger vor den DAC-Schnittstellen einer Sender-Implementierung. Seine Aufgabe ist zum einen das Anpassen der 16-Bit breiten, Vorzeichen behafteten Samples, zum anderen ermöglicht er es gegebenenfalls, bei mehreren SPS-Streams die parallel ankommenden Symbole umzusortieren, sodass die Samples der Symbole in richtiger Reihenfolge an die DACs gegeben werden.

Die Realisierung des Cyclic Prefix wird ebenfalls durch den Data Output Converter durchgeführt, indem er die Sendedaten entsprechend einliest: Zuerst wird die programmierte Anzahl an Cyclic Prefix Samples vom Ende des Symbols gelesen, anschließend das gesamte Symbol von Anfang bis Ende. Das Blockschaltbild des Converters in Abbildung 4.26 gibt eine Übersicht über die Struktur. Die Parallelität n wird dabei zur Designzeit festgelegt und hängt von der Anzahl der im System implementierten SPS-Streams ab.

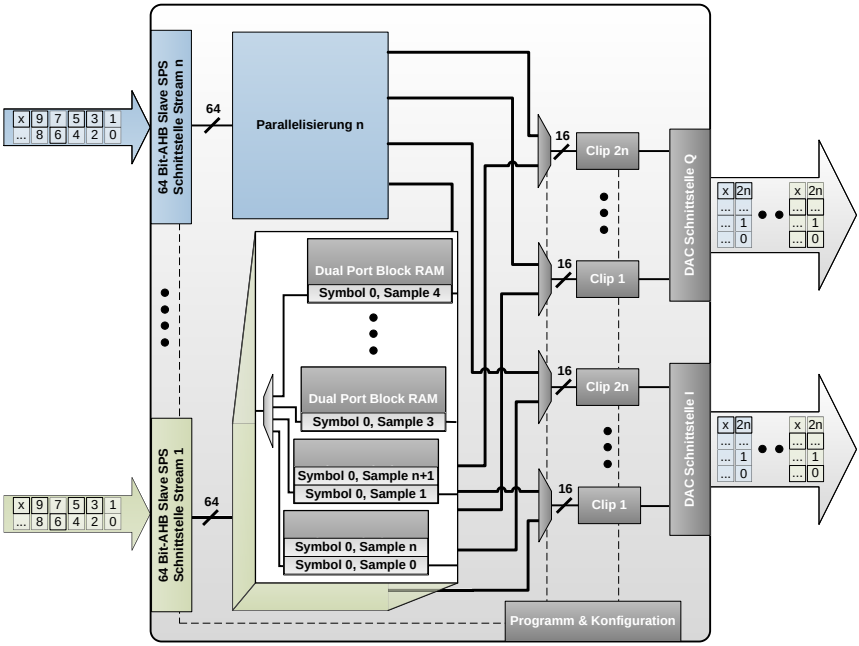


Abbildung 4.26.: Der Data Output Converter sortiert nicht nur die Ausgangsdaten richtig um, er führt auch ein Clipping und ein Runden auf die benötigte Auflösung für die Digital/Analog-Schnittstelle durch.

Der Data Output Converter erhält von jedem Stream pro Taktzyklus zwei Samples, welche jeweils zu einem anderen Symbol gehören. In den Parallelisierungssubmodulen findet die Konvertierung der Parallelität von Symbol-Ebene auf Sample-Ebene statt. Dazu besitzt jedes Submodul $2n$ 32-Bit Breite Block-RAMs, in welchen die Daten zwischengespeichert werden. Jedoch werden die Block-RAMs nicht nacheinander gefüllt, vielmehr wird für jedes Sample immer der nächste Block-RAM genommen, bis es nach Block-RAM $2n$ wieder mit dem ersten beginnt.

Auf diese Weise kann man die Block-RAM also als großen Speicher sehen, wobei die untersten Adress-Bits den zur Speicherung verwendeten Block-RAM auswählen. Sobald in jedem Parallelisierungsmodul genügend Samples beziehungsweise Symbole vorhanden sind, können über die zweiten Ports der Block-RAMs die

Samples parallel ausgelesen werden und zwar der Reihe nach, aus einem Parallelisierungsmodul nach dem andern.

Jeder der Blöcke zur *Parallelisierung* besitzt neben den vom SPS zugänglichen BRAM Blöcken zusätzlich einen Speicherbereich, in welchen der Zugriff auf den ersten Port nicht vom SPS aus, sondern vom CS aus erfolgt. In ihn lässt sich eine Präambel speichern, welche zu Beginn eines jeden OFDM-Blockes gesendet wird. Die Größe dieses Speicherbereichs ist zur Designzeit frei konfigurierbar.

Tabelle 4.10.: Syntheseergebnisse für unterschiedlich konfigurierte Data Output Converter für Virtex-6. DSP48E1 Blöcke werden nicht benötigt.

max. Anzahl	Ressourcentyp			Maximale
Streams	LUTs	Flip Flops	BRAMs	Taktfrequenz
1	1168	755	6	314 MHz
2	1947	1172	12	317 MHz
4	3076	1950	40	319 MHz
6	4028	2355	69	318 MHz

Bevor die Samples der DAC Schnittstelle übergeben werden, müssen sie in den insgesamt $4n$ Clip-Submodulen, so wie vom Programm vorgegeben, an das Format dieser Schnittstelle angepasst werden. Dabei kann nicht nur die Auflösung durch Begrenzen (Clipping), Runden und Erweitern beliebig geändert werden. Es ist auch möglich, das oberste Bit der Zahl zu invertieren, was von der verwendeten Zweierkomplement-Darstellung zu einer Repräsentation ohne Vorzeichen führt.

Der Ressourcenverbrauch des Converters hängt stark von der unterstützten Anzahl an SPS-Streams ab und ist in Tabelle 4.10 für verschiedene Stream-Konfigurationen zu sehen. Der Ressourcenverbrauch ist für eine maximale Größe der Präambel von 8192 komplexen Samples gelistet.

4.4.7.2. Synchronisationsbeschleuniger

Der Synchronisationsbeschleuniger ist wesentlich komplexer als der Data Output Converter und neben dem Equalization-Beschleuniger daher der zweite Grund, weshalb ein OFDM-Empfänger insgesamt aufwendiger ist und mehr Ressourcen benötigt als ein OFDM-Sender. Erste Untersuchungen wurden bereits in [Wöl12] durchgeführt, der hier vorgestellte Beschleuniger wurde allerdings von Grund auf neu entwickelt.

Wie in Abbildung 4.27 ersichtlich, besteht der Beschleuniger aus zwei Teilen. Der vordere Teil übernimmt die Synchronisierung sowohl im Zeitbereich als auch im

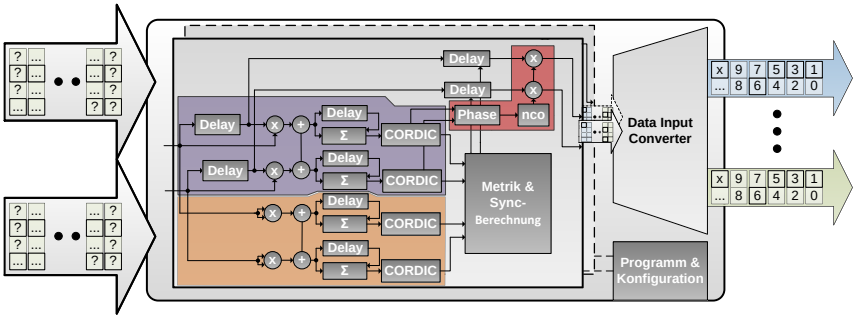


Abbildung 4.27.: Der Synchronisationsblock berechnet die benötigte Autokorrelation (violett) sowie die Energie (orange) des Eingangssignals, um eine Synchronisation in der Zeit zu finden. Des Weiteren kann optional eine Synchronisation im Frequenzbereich (rot) erreicht werden.

Frequenzbereich. Der hintere Teil (Data Input Converter) trennt den sequentiellen Strom symbolweise in parallele Streams und entfernt gleichzeitig den Cylic Prefix.

Für das Erkennen eines OFDM-Blocks und den zugehörigen Symbolgrenzen wird eine Autokorrelation sowie die Energie des Eingangssignals berechnet, vergleiche Kapitel 2.1.5. Die Berechnung der Autokorrelation, welche in Abbildung 4.27 violett hinterlegt ist, gleicht dabei bis auf den variablen Delay der orange hinterlegten Berechnung der Energie. Der nachfolgende Block zur Metrik- und Synchronisationsberechnung erlaubt den flexiblen gegenseitigen Vergleich der berechneten Werte, sowie den Vergleich mit programmierbaren Größen. Dazu können die errechneten Werte mit Hilfe des Programmcodes beliebig skaliert werden.

Anstatt die in Formel 2.22 aufgeführte Division zu implementieren, wird im Synchronisationsbeschleuniger die Autokorrelation von der Energie abgezogen, wodurch ebenfalls eine gewisse Normierung erreicht wird. Allerdings werden für diese Operation erheblich weniger Ressourcen benötigt. Abbildung 4.28 zeigt erstellte Aufnahmen echter OFDM Signale, welche mit Hilfe des On-Chip Logikanalysators Chipscope erstellt wurden.

Zur Metrik Analyse bietet der Synchronisationsbeschleuniger zwei Methoden, sowohl die Suche nach einer Spitze als auch die Suche nach einem Plateau. Eine Gegenüberstellung der Methoden ist in [63] zu finden. Zusammen mit verschiedenen Vergleichen zu Schwellenwerten lässt sich so eine Synchronisation im Zeitbereich für eine Vielzahl von unterschiedlichen Präambelvarianten berechnen.

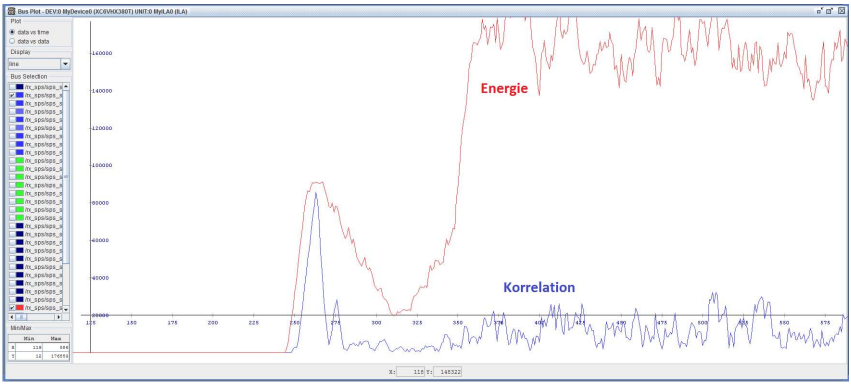


Abbildung 4.28.: Mit Chipscope aufgezeichneter Korrelations- und Energieverlauf echter empfangener OFDM Daten mit einer Präambel, welche eine Spitze als Korrelation erzeugt, vergleiche [63].

Des Weiteren kann neben der Synchronisation der Zeit auch eine Synchronisation der Frequenz stattfinden. Dazu wird das Signal durch einen digitalen IQ-Mischer mit einer Korrekturfrequenz gemischt, welche durch numerisch gesteuerte Oszillatoren erzeugt wird. Diese Korrekturfrequenz kann entweder statisch durch Software vorgegeben werden, oder aber sie wird aus der unterschiedlichen Phasenlage zweier sequentieller Präambel-Symbole berechnet, vergleiche Kapitel 2.1.5.2. Im Blockschaltbild sind die Komponenten hierfür rot hinterlegt.

Tabelle 4.11.: Virtex-6 Syntheseergebnisse für den Synchronisationsbeschleuniger mit unterschiedlichen Konfigurationen. Als max. Korrelationsfenster wurden 256 Samples gewählt.

max. Anzahl	Ressourcentyp				Maximale
Streams	LUTs	Flip Flops	BRAMs	DSP48E1	Taktfrequenz
1	12041	9503	80	24	104 MHz
2	24385	18159	160	42	104 MHz
4	53240	35516	336	78	104 MHz
6	88173	53153	528	122	104 MHz

Im Anschluss an die Synchronisierung werden die gültigen Samples eines jeden Symbols dem *Data Input Converter* gegeben, welcher die $2 - N$ parallelen Samples des sequentiellen Eingangsstroms in N Streams mit jeweils zwei Samples Parallelität konvertiert. Die Komplexität und der Ressourcenverbrauch des Synchroni-

sationsbeschleunigers hängen stark von N, der zur Designzeit bestimmten maximalen Anzahl an Streams, ab. In Tabelle 4.11 ist dies für verschiedene Konfigurationen dargestellt.

5. Experimente zur Framework basierten OFDM-Übertragung

Das folgende Kapitel beschreibt den Aufbau und die Ergebnisse der OFDM Experimente, welche im Rahmen dieser Arbeit durchgeführt wurden. Zunächst wird die Gestaltung und der Aufbau der einzelnen Experimente erläutert, wobei in Kapitel 5.1.1 näher auf die verwendete Prototyping Hardware eingegangen wird, die zur Analyse und Bewertung des entwickelten HW/SW Co-Design Frameworks verwendet wurde.

5.1. Aufbau der Experimente

Abbildung 5.1 zeigt den prinzipiellen Aufbau der Messungen von OFDM Signalen über eine Glasfaser, welche im Rahmen des in Kapitel 4.1 vorgestellten Szenarios benötigt werden. Der MPSoC OFDM Sender erzeugt ein digitales OFDM Signal im Basisband, das durch die Digital/Analog-Wandler in ein analoges OFDM Signal umgewandelt wird. Anschließend werden die In-phasen- und Quadraturkomponente mit einem analogen komplexen Mischer auf eine Zwischenfrequenz gemischt. Es resultiert ein Passbandsignal, mit welchem die Intensität einer Laserdiode (LD) moduliert wird.

Das optische Signal wird zum Empfänger über eine Einmodenfaser (Single-Mode Fibre, SMF) übertragen und dort mit einer Photodiode direkt empfangen. Der nachgeschaltete Analog/Digital-Wandler besitzt hierbei eine ausreichend große Bandbreite, um das Passbandsignal direkt in ein digitales Datensignal zu konvertieren. Ein Mischen in das Basisband erfolgt anschließend zusammen mit einer Reduktion der Samplerate durch einen Digital Down Converter (DDC). Das vom DDC ausgegebene IQ-Signal wird sodann durch den per Framework erstellten OFDM Empfänger demoduliert und ausgewertet.

Aufgrund mangelnder Verfügbarkeit entsprechender Hardware wurde das Experiment aus Abbildung 5.1, bei welchem alle DSP-Komponenten des Systems (Sender, Empfänger und Digital Down Converter) gleichzeitig in Echtzeit durch entsprechende FPGAs realisiert werden, in mehrere Experimente unterteilt. Diese Aufteilung in Sender-, Empfänger- und DDC-Experimente hat den Vorteil, dass

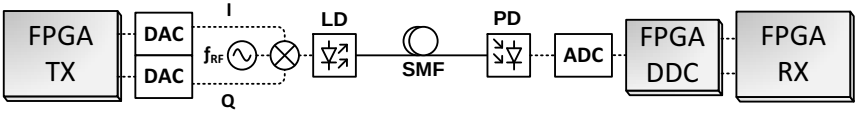


Abbildung 5.1.: Während am Sender ein analoger Mischer das Basisband auf die Zwischenfrequenz setzt, geschieht das Herabsetzen beim Empfänger mittels des Digital Down Converters digital im FPGA.

die Komponenten einzeln ausgemessen und ihre Eigenschaften somit genauer bestimmt werden können. Die Ergebnisse werden dabei aufgezeichnet und für weitere Experimente wiederverwendet. Dadurch kann besser nachvollzogen werden, welche Komponente welchen Einfluss auf die Qualität des Ergebnisses nimmt. In den folgenden Abschnitten werden die Sender- und Empfängerexperimente besprochen. Der Aufbau sowie die Experimente zum Digital Down Converter sind in Kapitel 6 thematisiert.

5.1.1. FPGA Prototyping Plattform und Analoge Schnittstellen

Zur Realisierung der HW/SW Co-Design basierten OFDM Systeme kommt als Hauptplatine ein *Xilinx Virtex 6 HXT 40G/100G Development Platform* Board der Firma Hitech Global zum Einsatz, welches mit einem XC6VHX380T-2FFG1923 Virtex-6 FPGA bestückt ist. Dieses bietet vier FPGA Mezzanine Card (FMC) Steckplätze, von denen alle GTX Transceiver bieten, jedoch nur zwei Stück auch mit Standard-IOs (insgesamt 132) belegt sind. Über die FMC Steckplätze lassen sich mit entsprechenden *8-Port SMA /34 Pairs LVDS FMC Module* Erweiterungskarten bis zu 32 GTX-Transceiver des FPGA auf *Sub-Miniature-A* (SMA) Buchsen herausführen.

Für die Empfangsseite kommen zwei verschiedene Typen Analog/Digital-Wandler zum Einsatz. Für das Konzept aus Abbildung 4.2 handelt es sich um einen *VEGA ADC30* der Firma MICRAM, welcher für die in Kapitel 6 beschriebenen DDC-Experimente eingesetzt wurde. Dieser Konverter bietet zwar nur eine Auflösung von 6-Bit, erreicht allerdings Sampleraten von bis zu 30 GSa/s. Der Wandler ist direkt kompatibel zu den GTX-Transceivern von Virtex-5 und Virtex-6 FPGAs, er wird über 24 dieser Transceiver angebunden.

Vega ADC30 Wandler sind nicht als einzelner Chip verfügbar, sondern sie sitzen auf kleinen Platinen. Alle Signale, bis auf den Takt und auf das analoge Signal, werden über spezielle Verbinder der Firma SAMTEC verbunden. Mit diesen können die Wandler auf eine Trägerplatine aufgesteckt werden. Um die Wandler mit

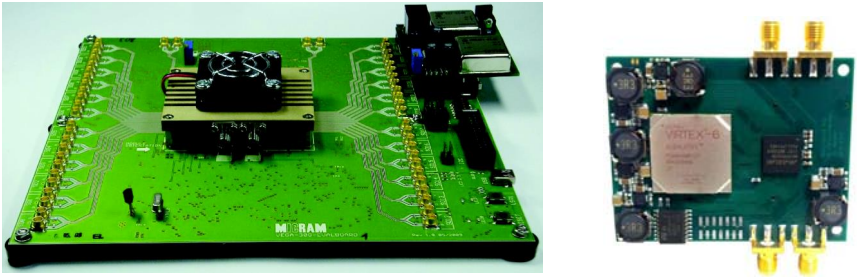


Abbildung 5.2.: Links ist ein MICRAM Evaluation Board mit aufgestecktem ADC30 Analog/Digital-Wandler von MICRAM zu sehen [60], rechts eine im Rahmen der Arbeit entstandene Adapterplatine, welche die Schnittstelle eines MAX5881 mit Hilfe eines kleinen Virtex-6 FPGAs auf die Schnittstelle eines ADC30 umsetzt.

Standard-SMA Steckverbinder ansprechen zu können kann ein *VEGA Evaluation Board* derselben Firma verwendet werden. Links im Bild 5.2 ist ein ADC30 Wandler auf ein solches Evaluationsboard aufgesteckt.

Um den OFDM Empfänger direkt, ohne vorgeschaltetes DDC-Design, zu bewerten, sind die Vega ADC30 Wandler ungeeignet. Zum einen kann nur einer dieser Wandler, aufgrund mangelnder GTX Transceiver, an die Hauptplatine angeschlossen werden, der Empfänger benötigt jedoch ein IQ-Signal. Zum anderen ist die hohe Samplerate dieser Wandler für den Empfang von Signalen im Basisband nicht sinnvoll einsetzbar, die geringe Auflösung ist jedoch von Nachteil. Abhilfe hierfür schaffen alternative ADC-Wandler des Typs *ADC12D1800* der Firma Texas Instruments. Sie bieten eine Samplerate von maximal 3,6 GSa/s bei 12-Bit Auflösung.

Ein ADC12D1800 Wandler gibt seine digitalen Daten mittels differenziellen Signalen und einem internen 1-zu-4 Demultiplexer aus. Um die Daten zweier Wandler entgegenzunehmen werden also alleine für die Datenverbindungen 192 IOs am FPGA benötigt. Da die verwendete Hauptentwicklungsplatine dem Anwender jedoch nur 132 solcher IOs des FPGAs zur Verfügung stellt, wurden Adapterplatinen entworfen, welche die Schnittstelle der Wandler auf jeweils 8 GTX-Verbindungen konvertieren.

Hierzu wurde für jeden Wandler eine Platine entworfen und hergestellt. Auf ihr steuert ein Virtex-6 FPGA des Typs XC6VLX75T-2FF484 die Wandler direkt an. Dabei wurde darauf geachtet, dass die Schnittstelle zur Hauptplatine kompatibel zur Schnittstelle der MICRAM VEGA ADC 30 ist, die Wandler also mit Hilfe der MICRAM Evaluierungsboards über SMA Steckverbinder angebunden wer-

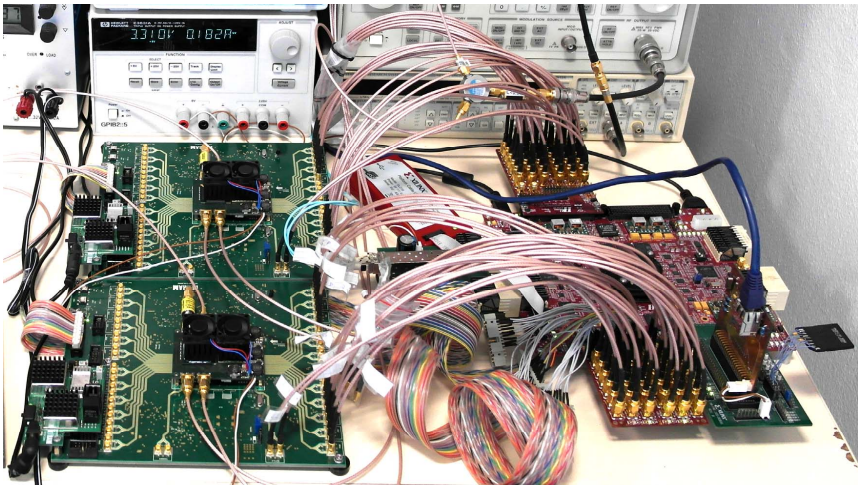


Abbildung 5.3.: Hardware des Empfängerprototyps: Zwei mit ADC12D1800 ADCs bestückte Adapterplatinen, aufgesteckt auf MICRAM Evaluationsboards und an die Hauptentwicklungsplatine per SMA-Kabel angebunden.

den können. Da hier allerdings nur 8 GTX-Transceiver pro Wandler benötigt werden, können zwei dieser ADCs auf diese Weise mit der Hauptplatine verbunden werden. Abbildung 5.3 zeigt einen Aufbau, in welchem entsprechend zwei dieser Wandler mittels Adapterplatinen und MICRAM Evaluierungsboards an die Hauptplatine angeschlossen sind.

Für den Prototypen des Senders werden *MAX5881* Digital/Analog-Wandler der Firma Maxim mit einer Auflösung von 12-Bit und einer maximalen Samplerate von 4,3 GSa/s verwendet. Sie werden, ähnlich den ADC12D1800, je mittels eines 4-fach Multiplexers über 48 LVDS Signale mit Sendedaten versorgt. Für diese Wandler wurden ebenfalls Adapterplatinen angefertigt, um sie über GTX-Transceiver an die Hauptplatine anzuschließen. Rechts im Bild 5.2 ist eine DAC-Adapterplatine vor Montage der Kühlkörper zu sehen. Die speziellen Verbinder der Firma SAMTEC zur Verbindung der Spannungsversorgungen und zur Verbindung der digitalen Datenleitungen befinden sich auf der Unterseite der Platine.

Die Schnittstelle zu den Adapterplatinen besteht somit aus einer GTX-Brücke zwischen dem FPGA der Hauptplatine und dem FPGA der Adapterplatinen. Das Blockschaltbild des Brückenmoduls ist in Abbildung 5.4 dargestellt. Das GTX-Brückenmodul besitzt vier verschiedene Taktdomänen, welche an den gestrichel-

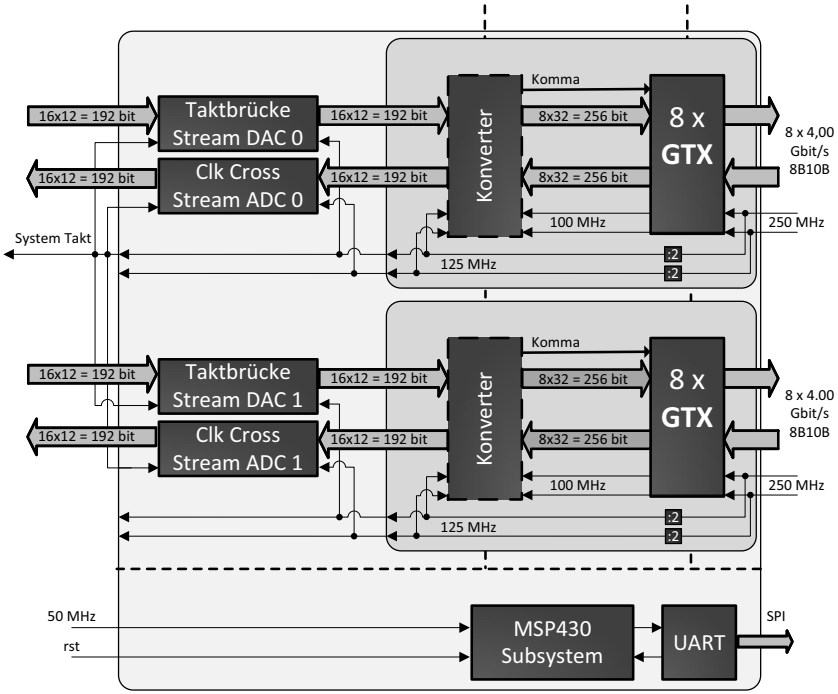


Abbildung 5.4.: Der Transport der Daten von der Hauptplatine zu den Wandlern sowie in entgegengesetzte Richtung geschieht über eine bidirektionale Datenbrücke aus GTX-Transceivern. Für Konfigurationsaufgaben enthält die Brücke ein Prozessor Subsystem.

ten Linien erkennbar sind. Die Konverter-Module beinhalten Logik zum Synchronisieren der einzelnen GTX-Transceiver sowie der GTX-Links zueinander.

Ein Prozessor-Subsystem, welches auf einem MSP430 kompatiblen Prozessor basiert, übernimmt dabei alle notwendigen Konfigurationen der Wandler sowie der GTX-Brücke. Der Prozessor, *openmsp430* genannt, steht unter der BSD-Lizenz und ist keine Eigenentwicklung, sondern stammt von Olivier Girard beziehungsweise von www.opencore.org, einer Webseite, die IP-Cores zur freien Verfügung bereitstellt, siehe [29].

Die Schnittstelle zum ADC30 Analog/Digital-Wandler von MICRAM ist ähnlich der GTX Brücke aus Abbildung 5.4 aufgebaut, allerdings werden hier 24 GTX Receiver benötigt, außerdem existieren Detailunterschiede in der Synchronisierung

der Links. Das Prozessor Subsystem kümmert sich um zusätzlich anfallende Kalibrierungsaufgaben.

Für detailliertere Informationen sei auf die Arbeitsergebnisse [25] von Herrn Dipl.-Ing. Michael Dreschmann innerhalb des Projektes *ACCORDANCE* verwiesen, in dessen Rahmen der Großteil der Adapterplatinen sowie der GTX-Brückenmodule zu selbigen entstanden sind.

5.2. Experimente mit dem programmierbaren OFDM Sender

5.2.1. FPGA Design des MPSoC Senders

Mit Hilfe des HW/SW Co-Design Frameworks wurden mehrere Sender-Varianten realisiert. Dabei wurde der Fokus für die Experimente auf eine 4-Stream Variante gelegt, da die Unterstützung von mehr als vier Streams beim Empfänger aus Ressourcengründen mit der zur Verfügung stehenden Hardware nicht möglich ist, siehe Kapitel 5.3. Eine grobe Übersicht der 4-Stream Variante ist als Blockschaltbild in Abbildung 5.5 dargestellt. Pro SPS-Stream werden zwei (i)FFT Beschleuniger realisiert, die maximale Transformationsgröße wurde auf 512-Stützstellen begrenzt. Die Größe der Dual Port AHB Speicher wurde auf 4096 komplexe Samples festgelegt, die maximale Anzahl an Pilottönen pro Symbol auf 64 Stück und die maximale Größe der Präambel auf 8192 komplexe Samples.

Tabelle 5.1.: Ressourcenverbrauch verschiedener Sender-Varianten. Prozentangaben beziehen sich auf die verfügbaren Ressourcen eines XC6VHX380T FPGAs.

Anzahl	Ressourcentyp								Maximale
Streams	LUTs		Flip Flops		BRAMs		DSP48E1		Taktfrequenz
2	61 108	25%	53 603	11%	168	21%	85	9%	100 MHz
4	92 654	39%	68 114	14%	276	36%	165	19%	100 MHz
8	174 029	72%	115 550	24%	540	70%	245	28%	80 MHz

Tabelle 5.1 zeigt den Ressourcenaufwand des aufgebauten Senders, sowie für Varianten mit zwei als auch mit acht SPS-Streams. Die Prozentangaben beziehen sich hier auf die in einem XC6VHX380T Virtex-6 FPGA verfügbaren Ressourcen, die angegebene Taktfrequenz ist die vorgegebene und von den Tools erreichte Frequenz. Allerdings konnten alle Designs ohne Probleme bei Experimenten um mindestens 25% übertaktet werden. Die Werte stammen aus den Berichten der Platzierung und Verdrahtung.

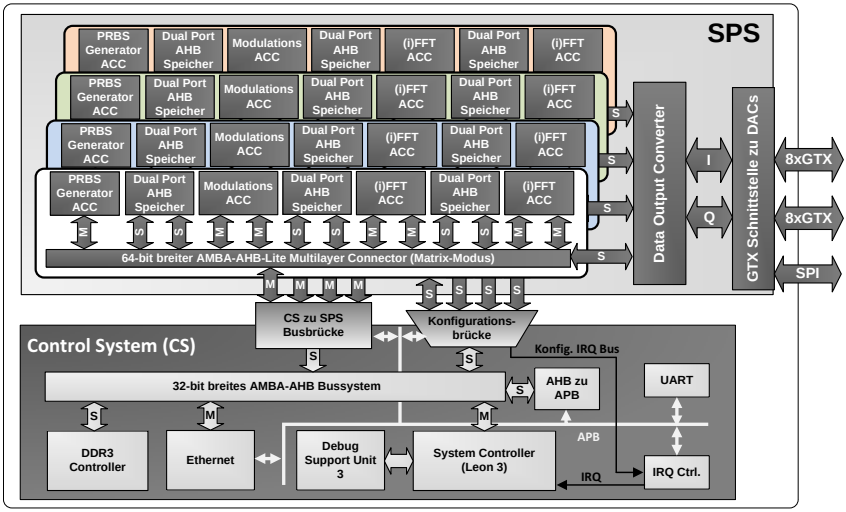


Abbildung 5.5.: Blockschaltbild eines Senders mit vier Streams, welcher durch ein auf 125 MHz übertaktetes SPS im Schnitt einen Durchsatz von 1 GSa/s erreicht.

Tabelle 5.2 zeigt eine feinere Verteilung des Ressourcenverbrauchs auf die einzelnen Komponenten. Die Werte dieser Auflistungen stammen aus den Berichten des Technologiemapping und können daher ebenso als hinreichend genau betrachtet werden. Prozessoren, die im Design mehrfach realisiert sind, besitzen je nach Platzierung und Verdrahtung der konkreten Instanz, minimale Unterschiede im Ressourcenverbrauch. Ihr Ressourcenbedarf wird daher in der Tabelle durch einen entsprechenden Mittelwert repräsentiert.

Es sei darauf hingewiesen, dass der Data Output Converter als 4-Stream Variante in der Tabelle gelistet ist. Die meisten Prozentangaben beziehen sich auf die verfügbaren Ressourcen des Chips der Hauptplatine, die Prozentangaben der SPS-Beschleuniger beziehen sich jedoch auf die Ressourcen des SPS, die Prozentangabe des OpenMSP430 Subsystems auf die Ressourcen der Analogschnittstelle.

Alle Angaben zum Ressourcenverbrauch sind in Abbildung 5.6 noch einmal anschaulich in Kuchendiagrammen dargestellt. Die oberen Diagramme zeigen, wie viele Ressourcen des Chips die Subsysteme des 4-Stream Designs jeweils benötigen. Es ist leicht zu sehen, dass das SPS zwar die meisten Ressourcen benötigt, allerdings die Ressourcen des Chips in der 4-Stream Variante noch nicht ausschöpft. Ebenfalls ist ersichtlich, dass in dieser Konfiguration Look-Up-Tables so-

Tabelle 5.2.: Ressourcenverbrauch des 4-SPS-Stream Senders, aufgeschlüsselt auf die einzelnen Komponenten.

Komponente	LUTs		Flip Flops		BRAMs		DSP48E1	
Control System[a]	17312	7%	13772	3%	28	4%	4	0%
CS zu SPS Brücke[a]	618	1%	421	1%	2	1%	0	0%
Konfig. Brücke[a]	518	1%	440	1%	2	1%	0	0%
Analogschnittstelle[a]	11736	5%	18083	4%	48	6%	1	0%
→OpenMSP430 Subsystem[b]	1529	13%	612	3%	4	8%	1	100%
Signal Proc. Sys.[a]	61929	0%	34546	7%	222	29%	160	19%
→AHB Connector (8x7)[c]	2809	5%	945	3%	0	0%	0	0%
→AHB Speicher (4096)[c]	26	0%	26	0%	4	1%	0	0%
→PRBS Generator Beschl.[c]	585	1%	516	1%	0	0%	0	0%
→Modulationsbeschl.[c]	1814	3%	982	3%	3	1%	0	0%
→(i)FFT Beschl. (512)[c]	4747	8%	2834	8%	15	7%	20	13%
→4 Stream Data Out. Conv.[c]	2300	4%	1349	4%	40	18%	0	0%

[a] Prozentangaben beziehen sich auf verfügbare Ressourcen eines XC6VHX380T
[b] Prozentangaben beziehen sich auf Ressourcen der Analogschnittstelle
[c] Prozentangaben beziehen sich auf Ressourcen des Signal Processing Systems

wie Block-RAMs zu ungefähr gleichen Teilen die kritischen Ressourcen darstellen.

In den unteren Diagrammen wird der Ressourcenverbrauch des SPS auf dessen Komponenten aufgeschlüsselt. Hieraus wird klar ersichtlich, dass der (i)FFT Beschleuniger die ressourcenintensivste Komponente des Senders darstellt, besonders, wenn mehr als ein Beschleuniger pro SPS-Stream realisiert wird. Man beachte außerdem, dass der Ressourcenverbrauch des Data Output Converters sich auf eine 4-Stream Variante bezieht. Um den Ressourcenverbrauch auf zwei Samples pro Takt zu normalisieren, müssen diese Angaben daher noch durch vier geteilt werden.

5.2.2. Versuchsaufbau zur Analyse des Senders

Bei den Experimenten mit dem MPSoC basierten OFDM Sender wurde zuerst das digitale Datensignal des Senders analysiert, um anschließend den Aufbau bis hin zu einer optischen Übertragung Schritt für Schritt zu erweitern. Abbildung 5.7 zeigt das komplette Setup, sowie die einzelnen Messpunkte A bis D. Diese Messpunkte zeigen, an welcher Stelle des Gesamtaufbaus das Signal mit einem 25 GSa/s Oszilloskop der Firma Tektronix aufgezeichnet wurde. Die aufgezeichneten Daten werden anschließend offline demoduliert.

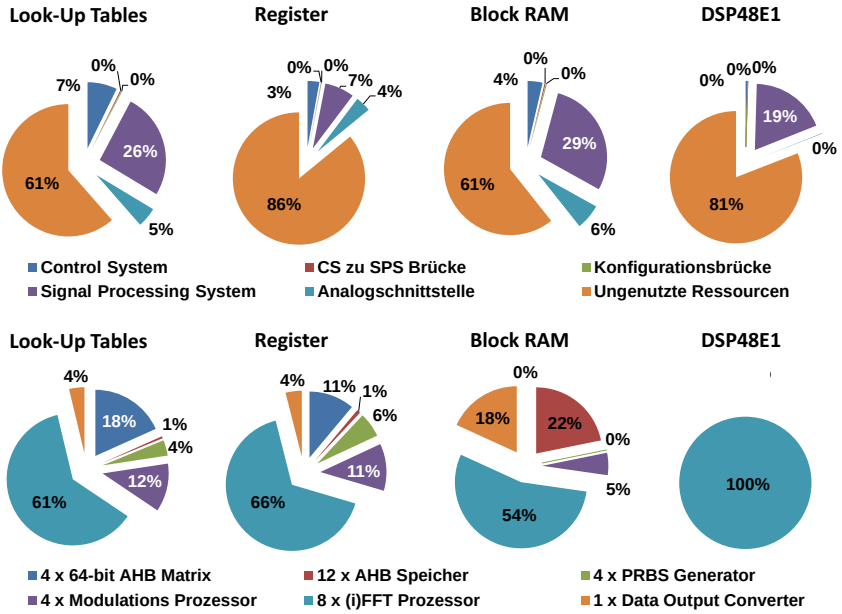


Abbildung 5.6.: Während die oberen Diagramme den Ressourcenverbrauch des Senders mit 4-Streams in Bezug auf die im Virtex-6 XC6VHX380T verfügbaren Ressourcen darstellen, wird in den unteren Diagrammen der Ressourcenverbrauch des SPS aufgeschlüsselt.

Zur Demodulation wurden die im Framework verwendeten Algorithmen in Matlab, einer kommerziellen Software zur Lösung mathematischer Probleme, erstellt. Allerdings rechnet Matlab standardmäßig mit einer Fließkommadarstellung. Ebenso wurden bestimmte Komponenten, welche innerhalb der Hardwarebeschleuniger zur Ressourcenoptimierung einfach gehalten wurden, durch genauere, vorgefertigte Matlab Funktionen ersetzt. Ein Beispiel hierfür sind die Tiefpassfilter der DDC oder die Interpolation der Korrekturfaktoren des Equalization-Beschleunigers.

Die Matlab Implementierung kann als Referenz für den Hardwareempfänger dienen, die EVM_M der demodulierten Werte in Matlab als obere Schranke für die demodulierten Werte des MPSoC Empfängers. Dadurch lässt sich bei der Analyse der Qualität des Sendesignals die Beeinträchtigung durch die Empfängerimple-

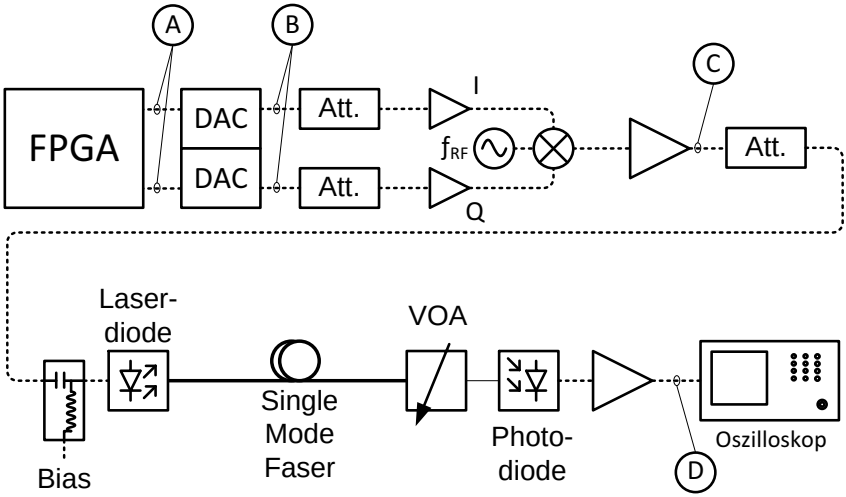


Abbildung 5.7.: Die Resultate des digitalen Signals aus Messpunkt A enthalten im Gegensatz zu Messungen an den DACs Ausgängen (B), Messungen nach dem Mischen auf eine Zwischenfrequenz (C), sowie Messungen einer optischen Übertragung (D), keine zusätzlichen Rausch- und Fehlerquellen.

mentierung reduzieren, um genauere Aussagen über die Qualität dieses Signals zu erhalten.

5.2.3. Messergebnisse der Sender-Experimente

Für die Analysen des Sendesignals wurden, ausgehend von einer Grundeinstellung, die wichtigsten OFDM-Parameter im Betrieb per Software geändert und das Ergebnis verglichen. Bei den Grundparametern handelt es sich um eine 64-Punkte FFT, wobei 48 Subträger mit Daten und vier mit Pilottönen moduliert werden, die 11 äußeren Träger sowie der DC-Träger bleiben unmoduliert. Die vier Pilottöne werden BPSK moduliert, die Datensymbole 16-QAM. Der Data Output Converter nimmt als Standardeinstellung für den Cyclic Prefix 16 Samples, dabei begrenzt er die Ausgabe nach oben um zwei Bit und rundet die untersten 2 Bit eines jeden 16 Bit Samples, wodurch man 12-Bit breite Samples als Ausgabe erhält.

In der Grundkonfiguration sendet der Empfänger einen OFDM Block nach dem anderen. Hierbei besitzt ein Block eine 800 Samples umfassende Präambel, wovon

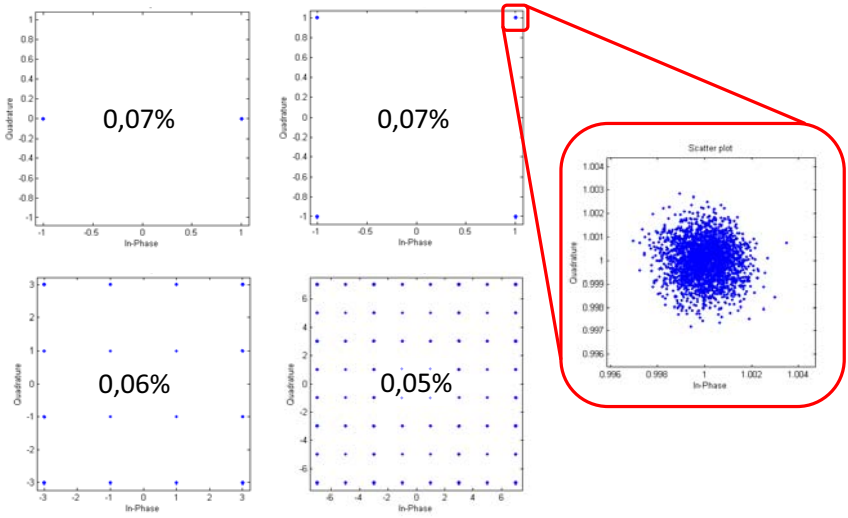


Abbildung 5.8.: Dargestellt sind die mittels Chipscope (Messpunkt A) erhaltenen digitalen Sendesignale für die Modulationsarten BPSK, QPSK, 16-QAM und 64-QAM, sowie ein stark vergrößerter Konstellationspunkt für QPSK.

die ersten 160 Samples die Präambel zur Synchronisation bilden. Bei den restlichen acht OFDM Symbolen sind jeweils acht Datenträger moduliert, zusammen modulieren sie nacheinander alle Träger mit einem Referenzwert und bilden somit die Trainingssequenz für den Equalizer. Nach der Präambel folgen im Block 448 Datensymbole, also 35840 Samples.

Als erster Parameter wurde die Modulationsart geändert und die Ergebnisse an Messpunkt A mittels des virtuellen Logikanalysators *Chipscope* direkt digital im FPGA abgegriffen. Die Daten beinhalten also lediglich Ungenauigkeiten der Architektur, welche beispielsweise auf Grund der begrenzten internen Zahlendarstellung entstehen. Abbildung 5.8 zeigt die Konstellationsdiagramme mit dem Wert der jeweiligen EVM_M, also der EVM, welche sich auf den längsten Vektor des Konstellationstyps bezieht.

Einen weiteren wichtigen Parameter stellt die Anzahl der Subträger und somit die FFT Größe dar. In Abbildung 5.9 sind die Spektren von OFDM Signalen mit 64, 128, 256 und 512 Subträgern mit jeweils 16 Samples Cyclic Prefix dargestellt. Für jede Konfiguration sind die Pilottöne, deren Anzahl mit zunehmender FFT-

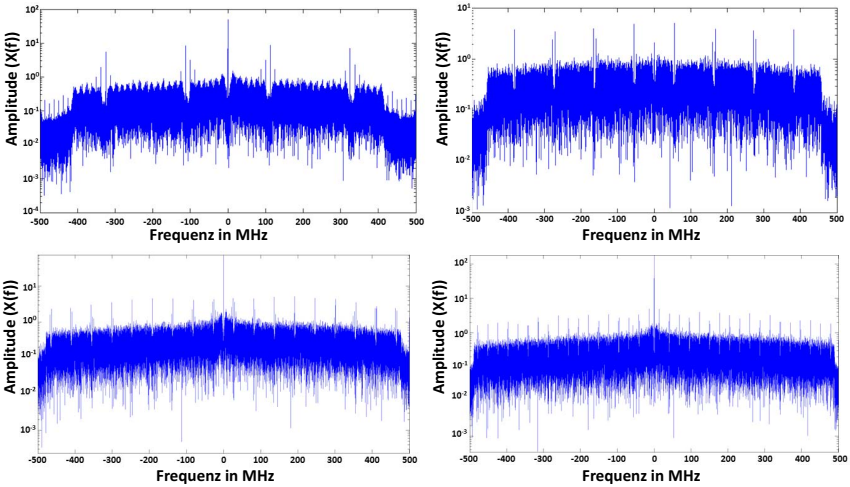


Abbildung 5.9.: Es werden die Spektren für verschiedene FFT-Größen gezeigt. Oben links: 64, oben rechts: 128, unten links: 256 und unten rechts: 512.

Größe auch erhöht wurde, leicht erkennbar. Auch gut zu erkennen ist, dass die Zahl der unmodulierten Träger am Rande der Spektren beibehalten wurde. Da für 64 Subträger ein größerer Trägerabstand vorliegt, nimmt der unmodulierte Bereich am Rand für diese Variante am meisten Bandbreite ein.

Tabelle 5.3 gibt einen Überblick über die Parameter sowie über die erreichten Datenraten. Die Messungen sind dabei nach Subträgeranzahl gruppiert. Die Werte der Datenraten lassen sich mit Hilfe von Formel 5.1 berechnen. Hier werden weder die Präambel noch die Pilotträger oder unmodulierte Träger betrachtet. Auch der Cyclic Prefix muss herausgerechnet werden.

$$\text{Datenrate} = \frac{(\#DC \cdot \#SpF)}{(\#SC + \#CP) \cdot \#SpF + \#SpP} \cdot SF \cdot ld(M) \tag{5.1}$$

mit

$$\begin{aligned}
\#SC &= \text{Anzahl Subträger} \\
\#DC &= \text{Anzahl Datenträger} \\
\#CP &= \text{Sampleanzahl des Cyclic Prefix} \\
\#SpF &= \text{Anzahl an OFDM-Symbolen pro OFDM-Frame} \\
\#SpP &= \text{Anzahl an Samples pro OFDM-Präambel} \\
M &= \text{Wertigkeit des Modulationsformats} \\
SF &= \text{Samplingfrequenz}
\end{aligned}$$

Tabelle 5.3.: Parameter der Senderexperimente sowie die erreichten Datenraten. Die Parameter beziehen sich jeweils auf einen OFDM-Frame.

Subträger (#SC)	**64**	**128**	**256**	**512**
Samplingfrequenz [SF]	1 GHz	1 GHz	1 GHz	1 GHz
Anzahl SPS-Streams	4	4	4	4
SPS-Taktfrequenz	125 MHz	125 MHz	125 MHz	125 MHz
Unmodulierte Träger	12	12	12	12
Datenträger (#DC)	48	108	226	464
Pilotträger	4	8	18	36
Cyclic Prefix [#CP] in Sa.	16	16	16	16
Sync-Präambel in Sa.	160	160	160	160
EQ-Präambel in Sym.	8	8	8	16
EQ-Präambel in Sa.	640	1152	2176	8448
Präambel [#SpP] in Sa.	800	1312	2336	8608
Datensymbole [#SpF] in Sym.	448	224	112	56
Symboldauer	80 ns	144 ns	272 ns	528 ns
Datenphase	35,84 µs	32,26 µs	31,58 µs	29,57 µs
Präambelphase	0,8 µs	1,3 µs	2,2 µs	8,6 µs
Framedauer	36,6 µs	33,6 µs	32,8 µs	38,2 µs
Datenrate **BPSK** (M=2)	0,59 Gb/s	0,72 Gb/s	0,77 Gb/s	0,68 Gb/s
Datenrate **QPSK** (M=4)	1,17 Gb/s	1,44 Gb/s	1,54 Gb/s	1,36 Gb/s
Datenrate **16-QAM** (M=16)	2,35 Gb/s	2,88 Gb/s	3,09 Gb/s	2,72 Gb/s
Datenrate **64-QAM** (M=64)	4,70 Gb/s	5,76 Gb/s	6,17 Gb/s	5,45 Gb/s

Man sieht, dass zunächst mit der Subträgeranzahl die Datenrate steigt. Der Hauptgrund hierfür besteht im geringeren prozentualen Anteil des Cyclic Prefix, bezogen auf ein Symbol. Zum anderen verschiebt sich das Verhältnis von modulier-

ten Trägern, unmodulierten Trägern und Pilotträgern zugunsten der modulierten Träger.

Diesem Effekt wirkt entgegen, dass mit der FFT-Größe ebenfalls die Größe der Trainingssequenz zur Kanalschätzung steigt (EQ-Präambel). Somit steigt die Dauer der Präambel-Phase, die Daten-Phase des OFDM Frames bleibt jedoch unverändert, daher sinkt die Datenrate. Für das 512 Subträger System, in welchem die EQ-Präambel auf 16 anstatt auf 8 Symbole verteilt wurde, ist die Datenrate daher sogar geringer als für das 256 Subträger System. Durch unterschiedliche Einstellungen der Präambel, des Cyclic Prefix sowie der Anzahl an Datenträgern lassen sich noch viele weitere, auch höhere Werte erreichen.

Die erreichten Datenraten liegen im angestrebten Gb/s-Bereich. Die Datenraten der restlichen Sender-Implementierungen aus Tabelle 5.1 sind in den Tabellen 5.4 und 5.5 zu sehen. Insgesamt konnten für die verfügbare Prototyping Plattform somit Sender-Varianten realisiert werden, welche den gesamten einstelligen Gigabit-Bereich abdecken.

Tabelle 5.4.: Datenraten des 2-SPS-Stream Designs.

2 SPS-Stream-Design 125 MHz

Subträger (#SC)	**64**	**128**	**256**	**512**
Sampling Frequenz [SF]	500 MHz	500 MHz	500 MHz	500 MHz
Datenrate **BPSK** (M=2)	0,29 Gb/s	0,36 Gb/s	0,39 Gb/s	0,34 Gb/s
Datenrate **QPSK** (M=4)	0,59 Gb/s	0,72 Gb/s	0,77 Gb/s	0,68 Gb/s
Datenrate **16-QAM** (M=16)	1,17 Gb/s	1,44 Gb/s	1,54 Gb/s	1,36 Gb/s
Datenrate **64-QAM** (M=64)	2,35 Gb/s	2,88 Gb/s	3,09 Gb/s	2,72 Gb/s

Tabelle 5.5.: Datenraten des 8-SPS-Stream Designs.

8 SPS-Stream-Design 110 MHz

Subträger (#SC)	**64**	**128**	**256**	**512**
Sampling Frequenz [SF]	1,76 GHz	1,76 GHz	1,76 GHz	1,76 GHz
Datenrate **BPSK** (M=2)	1,03 Gb/s	1,27 Gb/s	1,36 Gb/s	1,2 Gb/s
Datenrate **QPSK** (M=4)	2,07 Gb/s	2,54 Gb/s	2,72 Gb/s	2,4 Gb/s
Datenrate **16-QAM** (M=16)	4,13 Gb/s	5,07 Gb/s	5,43 Gb/s	4,79 Gb/s
Datenrate **64-QAM** (M=64)	8,26 Gb/s	10,15 Gb/s	10,87 Gb/s	9,58 Gb/s

Die Messungen an den Messpunkten B, C und D aus Abbildung 5.7 wurden wieder mit den Grundparametern durchgeführt, wobei die Subträger einmal mit QPSK und einmal mit einer 16-QAM moduliert wurden. Abbildung 5.10 zeigt die durch Matlab berechneten Konstellationsdiagramme der gemessenen Signale.

Die ersten Messungen an Messpunkt B betrachten das durch einen elektrischen Mischer auf eine Zwischenfrequenz von 5 GHz hochgesetzte Signal. Mit Hilfe des Oszilloskops wurde ein Frame aufgenommen, mit einem in Matlab realisierten Digital Down Converter ins Basisband gemischt und anschließend wieder mit Matlab demoduliert. Die erreichte Größe des Fehlervektors ist 2,72% für QPSK und 2,21% für 16-QAM und bezieht sich auch hier wieder auf den größten Vektor des jeweiligen Modulationsformates.

Im Vergleich zu den Messungen aus Messpunkt A resultiert die höhere EVM_M zum einen aus Ungenauigkeiten der DACs und den ADCs des Oszilloskops. Zum anderen können hier und in den folgenden Messungen auch Abweichungen in den anvisierten Zwischen- und Samplingfrequenzen entstehen, welche zwar teilweise erkannt und korrigiert werden, jedoch nicht vollständig zu vermeiden sind. Die Messungen an Messpunkt C besitzen bereits eine EVM_M von 4,3% für QPSK und 4,53% für 16-QAM, was hauptsächlich auf das zusätzliche Rauschen des Mischers zurückzuführen ist.

Die letzten in Abbildung 5.10 gezeigten Ergebnisse, welche eine vollständige optische Übertragung darstellen, erreichen eine EVM_M von 7,78% für QPSK und 7,94% für 16-QAM. Insgesamt liegen alle Ergebnisse also unter dem Schwellenwert dessen, was man mit dem Einsatz moderner Forward Error Correction (FEC) für eine fehlerfreie Übertragung tolerieren kann, vergleiche Kapitel 2.1.7.

5.3. Experimente mit dem programmierbaren OFDM Empfänger

5.3.1. FPGA Design des MPSoC Empfängers

Ähnlich dem OFDM-Sender Prototypen wurde auch ein OFDM-Empfänger mit Hilfe des HW/SW Co-Design Frameworks in Hardware aufgebaut. Mit ihm sollten nicht nur die Funktionen und die Möglichkeiten der übrigen Beschleuniger getestet werden, durch das Auswerten und den Vergleich von Referenz-Sendedaten und den experimentellen Sendedaten sollten ebenfalls Aussagen über ein Übertragungssystem getroffen werden, welches komplett auf dem Framework basiert.

Für den OFDM-Empfänger kommen anstelle der Modulationsbeschleuniger Demodulationsbeschleuniger zum Einsatz, der Data Output Converter wird durch den Synchronisationsbeschleuniger ersetzt und zusätzlich sind Equalization-Beschleuniger sowie ein weiterer AHB Speicher pro Stream in das Design integriert. Auch die GTX Schnittstellen zu den Analogwandlern sind leicht angepasst. So werden nun die GTX-Empfänger innerhalb der Schnittstelle verwendet und das

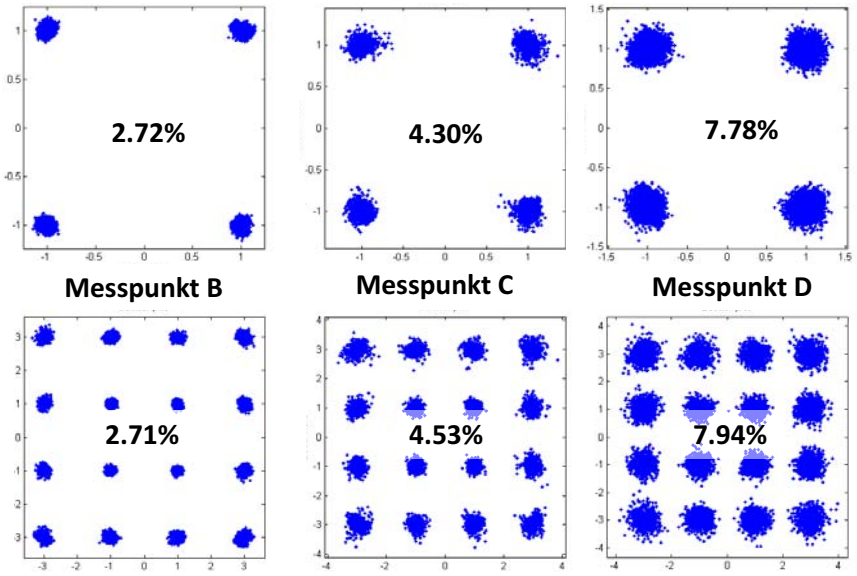


Abbildung 5.10.: Die Resultate der Messungen an den DACs Ausgängen (B), Messungen nach dem Mischen auf eine Zwischenfrequenz (C) sowie Messungen einer optischen Übertragung (D) enthalten im Vergleich zum digitalen Signal (A) zusätzliche Rausch- und Fehlerquellen.

Programm des Prozessorsubsystems ist für die ADCs von Texas Instruments angepasst. Abbildung 5.11 veranschaulicht den Aufbau des SPS, das CS bleibt unverändert zum Design des Senders.

Der Vergleich einer 2-Stream und einer 4-Stream Variante in Bezug auf den Ressourcenverbrauch ist in Tabelle 5.6 zu sehen. Während bei der 4-Stream Variante noch reichlich Ressourcen des FPGAs ungenutzt sind, befindet sich ein 4-Stream Empfänger bereits am äußersten Limit der Ressourcen eines XC6VHX380T Virtex-6 FPGA. Dementsprechend wurde keine 8-Stream Variante eines Empfängers erstellt.

Eine Übersicht über den aufgeschlüsselten Ressourcenverbrauch der 4-Stream Empfängerimplementierung findet sich in Tabelle 5.7. Die Werte des Synchronisationsbeschleunigers stehen für eine 4-Stream Variante, die anderen Ressourcen beziehen sich auf jeweils eine Instanz eines Beschleunigers. Alle Einstellungen,

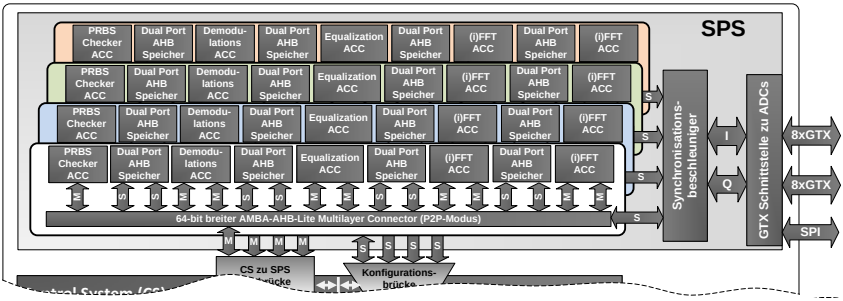


Abbildung 5.11.: Der Empfängerdesignprototyp wurde ebenfalls mit 4 SPS-Streams realisiert, was bei 125 MHz Taktfrequenz einem Durchsatz von 1 GSa/s entspricht.

Tabelle 5.6.: Ressourcenverbrauch verschiedener Empfänger-Varianten. Prozentangaben beziehen sich auf die verfügbaren Ressourcen eines XC6VHX380T FPGAs.

Anzahl	Ressourcentyp								Maximale
Streams	LUTs		Flip Flops		BRAMs		DSP48E1		Taktfrequenz
2	130 149	54%	111 202	23%	364	47%	247	28%	100 MHz
4	201 508	84%	160 506	34%	648	84%	483	56%	100 MHz

die zur Designzeit benötigt werden (beispielsweise maximale FFT-Größe), wurden auf die Werte des Sender-Prototyps festgelegt.

Man erkennt, dass das SPS drei Viertel der Ressourcen des FPGAs belegt. Mehr als ein Viertel dieser SPS-Ressourcen benötigt hierbei der Synchronisationsbeschleuniger. Allerdings wird dieser im Design nur einmal verwendet und sollte zur Normalisierung durch die Anzahl an SPS-Streams, in diesem Fall durch vier, geteilt werden.

Der Equalization-Beschleuniger verwendet mehr als 10 Prozent der SPS-LUTs und der SPS-Register. Wenn man die Equalization-Beschleuniger aller Streams zusammen nimmt, belegen diese somit nahezu die Hälfte der gesamten SPS-Ressourcen. Somit verwendet dieser Beschleuniger eindeutig die meisten Ressourcen. Bedenkt man, dass er nur in OFDM-Empfängern benötigt wird, ist auch leicht nachvollziehbar, aus welchem Grund der Empfängerprototyp wesentlich mehr Ressourcen verbraucht.

Die Kuchendiagramme in Abbildung 5.12 stellen die Aufteilung der Ressourcen für das Gesamtsystem sowie für das SPS dar. Gerade das Aufschlüsseln auf SPS-

Tabelle 5.7.: Aufgeschlüsselter Ressourcenverbrauch des Empfängers.

Komponente	LUTs		Flip Flops		BRAMs		DSP48E1	
Control System[a]	16210	7%	13274	3%	28	4%	4	0%
CS zu SPS Brücke[a]	109	0%	155	0%	2	0%	0	0%
Konfig. Brücke[a]	590	0%	440	0%	2	0%	0	0%
Analogschnittstelle[a]	11760	5%	17807	4%	48	6%	1	0%
→OpenMSP430 Subsys.[b]	1495	13%	612	3%	4	8%	1	100%
Signal Proc. Sys.[a]	173338	73%	125839	26%	582	76%	478	55%
→AHB Connector (P2P)[c]	18	0%	18	0%	0	0%	0	0%
→AHB Speicher (4096)[c]	13	0%	13	0%	4	1%	0	0%
→PRBS Checker Besch.[c]	610	0%	467	0%	0	0%	0	0%
→Demodulationsbeschl.[c]	846	0%	742	1%	1	0%	0	0%
→(i)FFT Beschl. (512)[c]	5050	3%	2944	2%	15	3%	20	4%
→Equalization-Beschl.[c]	19508	11%	15379	12%	12	2%	60	13%
→4-Stream Sync. Beschl.[c]	48301	28%	35215	28%	336	58%	78	16%

[a] Prozentangaben beziehen sich auf verfügbare Ressourcen des XC6VHX380T
[b] Prozentangaben beziehen sich auf Ressourcen der Analogschnittstelle
[c] Prozentangaben beziehen sich auf Ressourcen des SPS

Beschleuniger zeigt sehr gut den Mehraufwand, der für Empfänger durch Kanalkorrektur und Synchronisation gegeben ist.

5.3.2. Versuchsaufbau der Empfängerexperimente

Zuerst wurden, wie schon bei den Experimenten zum Sender, digitale Experimente durchgeführt. Dies ermöglichte das isolierte Testen des MPSoC Designs, wodurch die Ergebnisse frei von den spezifischen Rauschquellen der verwendeten analogen Hardware und der Messstrecke sind.

Für die Durchführung wurde das FPGA Design des Empfängers leicht erweitert und die digitalen Daten der Sender-Experimente aus Messpunkt A direkt in noch unbelegte BRAM geschrieben. Die digitalen Ausgangsdaten des Senders wurden auf 12 Bit Auflösung reduziert und direkt dem Synchronisationsbeschleuniger zur Verfügung gestellt. Nach einer erfolgreichen Synchronisation wurden die Daten planmäßig vom OFDM-Empfänger demoduliert. Für die Experimente wurden wieder die selben unterschiedlichen Parameter-Einstellungen verwendet, wie sie schon bei den Sender-Experimenten zum Einsatz kamen. Die demodulierten Daten wurden mit Hilfe von Chipscope hinter den Demodulationsbeschleunigern ausgelesen, um anschließend mittels Matlab die EVM_M zu errechnen.

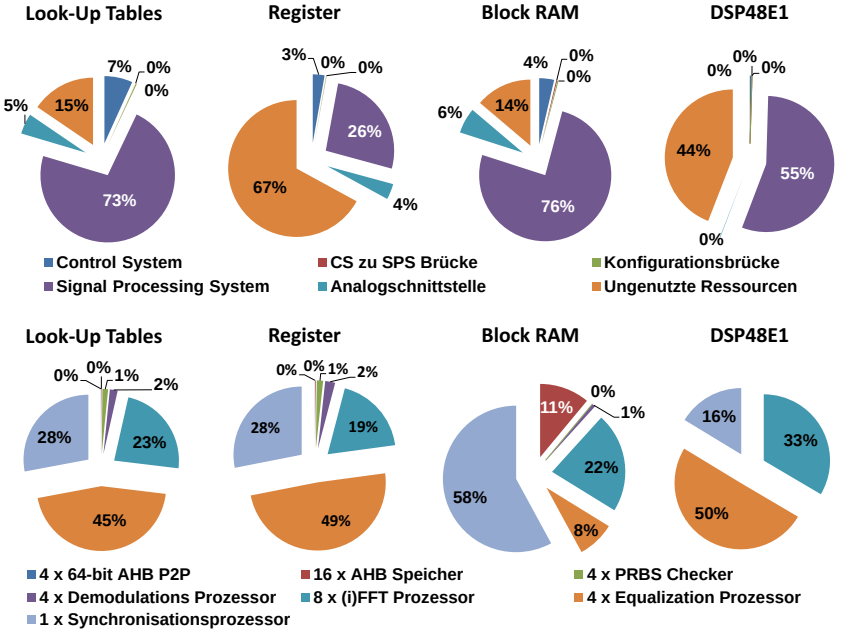


Abbildung 5.12.: Oben befinden sich die Kuchendiagramme, welche den Ressourcenverbrauch des 4-Stream Empfängers in Bezug auf die im Virtex-6 XC6VHX380T verfügbaren Ressourcen wiedergeben, die unteren Diagramme zeigen den aufgeschlüsselten Ressourcenverbrauch des SPS.

Um neben den rein digitalen Experimenten auch Experimente mit einer echten Übertragung zu testen, wurde ein AWG der Firma Tektronix verwendet, welcher über die in Anhang A.2 vorgestellten Tiefpassfilter an den Empfängerprototypen angeschlossen wurde, siehe Abbildung 5.13. Dazu wurden verschiedene Datensätze in den AWG geladen und dann mit der entsprechenden Samplerate abgespielt. Die demodulierten Daten wurden mit Hilfe von Chipscope aus dem FPGA extrahiert und das Konstellationsdiagramm sowie die EVM_m mit Matlab berechnet.

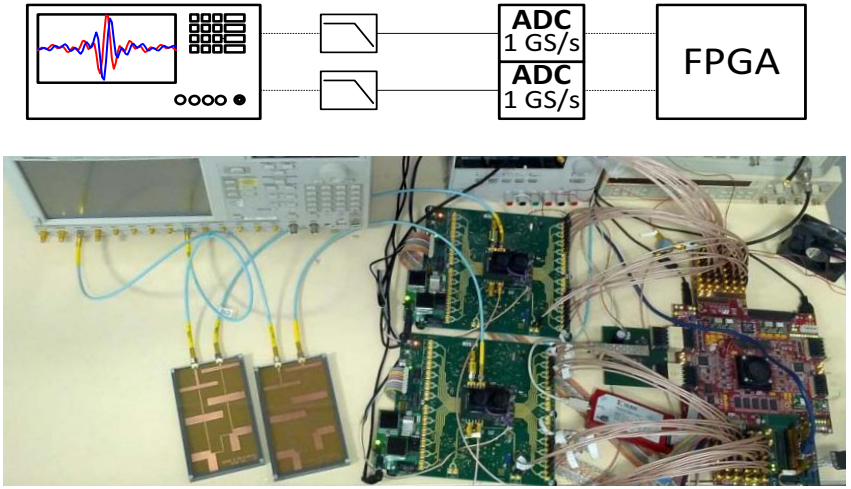


Abbildung 5.13.: Versuchsaufbau der Empfängermessungen: Über Tiefpässe wurde ein AWG direkt an die ADCs des Empfängers angeschlossen.

5.3.3. Ergebnisse der Empfängermessungen

Abbildung 5.14 zeigt die EVM_m Werte der demodulierten Daten für die rein digitalen Experimente und vergleicht sie mit den Ergebnissen der Sender-Experimente. Dabei kamen wieder die verschiedensten Kombinationen von FFT-Größe und Modulationsart der einzelnen Subträger zum Einsatz. Für alle anderen Einstellungen wurden die Grundparameter verwendet. Eine Auflistung der wichtigsten OFDM Parameter sowie der erzielten Datenraten ist durch Tabelle 5.3 gegeben. Da der Empfänger das Signal des durch diese Tabelle repräsentierten Senders empfängt, gelten die Werte dieser Tabelle auch für ihn. Gleiches gilt für Tabelle 5.4 und die 2-SPS-Stream Variante des Empfängers.

Die Summe der Ungenauigkeiten im Sender und im Empfänger befinden sich für jede Kombination von Parametern in einem Rahmen, welcher eine EVM_m von weniger als 0,5% ermöglicht. Dabei ist der Anteil der Degradierung durch den Empfänger größer, da sich bereits bestehende Ungenauigkeiten des Senders durch den Equalizer im Empfänger verstärken und auf alle Werte übertragen.

Die Ergebnisse der Empfängerexperimente aus Abbildung 5.13 sind in Abbildung 5.15 dargestellt. Die oberen Bilder zeigen dabei die Sendesignale, also die Daten, mit welchen der AWG geladen wurde. Sie stammen allesamt aus den Messungen

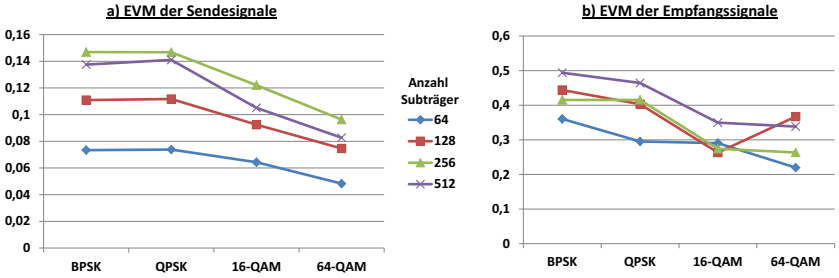


Abbildung 5.14.: Vergleich der EVM_M Werte der Sendesignale und der Empfangssignale der digitalen Experimente für verschiedene Kombinationen von der Anzahl an Subträgern und der Modulationsart.

der Sender-Experimente aus Kapitel 5.2. Die untere Hälfte zeigt die entsprechenden Ergebnisse der Demodulation durch den Empfänger Prototypen in Form der Konstellationsdiagramme und der EVM mit Bezug auf den maximalen Vektor der Modulationsart.

Zuerst wurden, ganz links im Bild, die digitalen Daten des Messpunktes A verwendet. Im Vergleich zur Referenz Demodulierung durch Matlab, welche eine EVM_M von lediglich 0.07% liefert, demoduliert der MPSoC Empfänger das Signal mit 2.07%. Allerdings liegt dies, wie die vorhergehenden digitalen Experimente zeigen, weniger am MPSoC sondern hauptsächlich an den analogen Rauschquellen, in diesem Fall an den Digital/Analog-Wandlern des AWG und den Analog/-Digital-Wandlern des Empfängerprototyps.

Für die Sendesignale von Messpunkt B und somit quasi für direkte Verbindung der DACs des MPSoC-Senders mit den ADCs des MPSoC-Empfängers, erhöht sich die EVM_M durch den Realtime-Empfänger im Vergleich zur Matlab Demodulierung von 2,71% auf 3,19%.

Bei der Verwendung der Werte aus Messpunkt C muss das Signal, welches durch den analogen elektrischen Mischer auf eine Zwischenfrequenz gehoben wurde, wieder ins Basisband gebracht werden, damit der MPSoC Empfänger dieses ohne Weiteres demodulieren kann. In diesen Versuchsreihen geschieht dies digital in Matlab, bevor die Daten in den AWG geladen werden. Eine Echtzeitimplementierung mit Hilfe eines Digital Down Converters wird in Kapitel 6.1 untersucht. Beim Empfang dieser Übertragung steigt die EVM_M von 4,53% auf 5,08% im Vergleich zur Referenz.

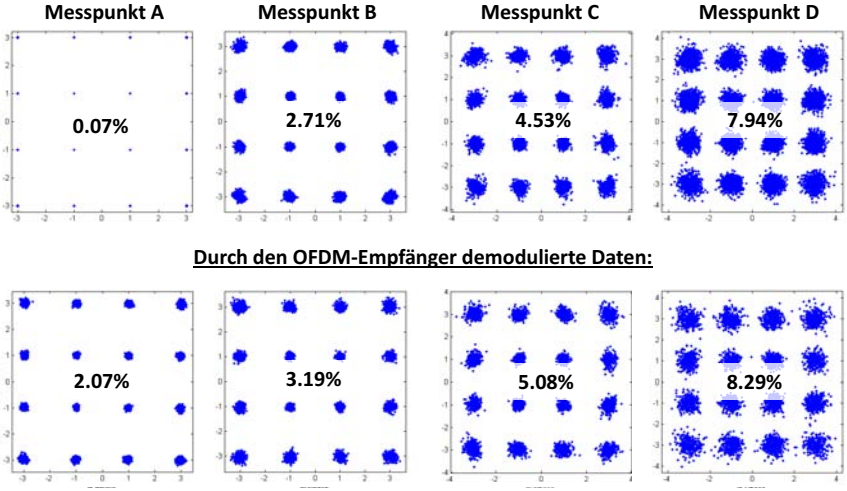


Abbildung 5.15.: Während die Konstellationsdiagramme oben die Sendesignale zeigen, also die Signale, mit denen der AWG geladen wurde, werden in der unteren Bildhälfte die Ergebnisse der Demodulation durch den MPSoC Empfänger dargestellt.

Beim letzten Experiment der Versuchsreihe, der optischen Übertragung mit den Daten von Messpunkt D, erzeugt der MPSoC Empfänger einen Anstieg der EVM_M von 7,94% auf 8,29%. Auch hier wurde die DDC Funktionalität durch Matlab realisiert. Alle Versuche, auch die optische Übertragung, erreichten auch hier EVM_M Werte, welche mit entsprechenden FEC-Implementierungen als hinreichend fehlerfrei angesehen werden können, siehe Kapitel 2.1.7.

5.4. Vergleich mit bereits publizierten OFDM-Systemen

Durch das Ziel des Frameworks, eine Lücke zwischen Durchsatz und Flexibilität zu schließen, fällt ein wertender Vergleich mit existierenden Architekturen schwierig, da kein System mit derselben Ausrichtung existiert. Die meisten auf Durchsatz optimierten OFDM Systeme nutzen dedizierte Schaltungen, welche in FPGAs implementiert wurden. Diese Systeme bieten in der Regel eine höhere Bitrate als die hier aufgebauten Prototypen, meist lässt sich damit allerdings höchstens die Modulationsart der einzelnen Subträger ändern.

Der Vergleich mit Systemen, welche sich durch Software konfigurieren lassen, führt immer zu dem Ergebnis, dass diese wesentlich niedrigere Datenraten besitzen. So können weder die GPUs, die DSP-Realisierungen, noch die MPSoCs mit Hardwarebeschleunigern in den Gigabitbereich vorstoßen. Sie sind allerdings auch nie auf Durchsatz optimiert, ihre Ziele bestehen meist darin, den Energieverbrauch und die Kosten für einen spezifischen Standard-Anwendungsfall zu senken.

Tabelle 5.8.: Vergleich der Eigenschaften der aufgebauten Prototypen mit publizierten OFDM-Signalverarbeitungssystemen.

	Diese Arbeit	**Yeh et al. [117]**	**Yan et al. [116]**	**Ma et al. [57]**
Jahr	2014	2010	2010	2013
Architektur	MPSoC	DSP	DSP	GPU
Baustein	Virtex-6	TMS320C5409	TMS320C6000	GF580GTX
Anmerkung	-	IEEE 802.11n	MIMO-OFDM	-
Modulation	64-QAM	64-QAM	QPSK	1024-QAM
Max. Samplerate	1 GSa/s	313 MSa/s	48 KSa/s	192 MSa/s
Max. Bitrate	10 Gb/s	600 Mb/s	6,4 Kb/s	476 Mb/s
Flexibilität	sehr hoch	voll	voll	voll

	Diese Arbeit	**Harju [35]**	**Qian et al. [75]**	**Schmogrow et al. [SWN+11]**
Jahr	2014	2006	2010	2011
Architektur	MPSoC	MPSoC	Dediziert	Dediziert
Baustein	Virtex-6	ASIC	16 x Virtex-5	Virtex-5
Anmerkung	-	3 Beschleuniger	Eine Taktquelle	nur Sender
Max. Modulation	64-QAM	-	QPSK	16-QAM
Max. Samplerate	1 GSa/s	20-80 MSa/s	20 GSa/s	28 GSa/s
Max. Bitrate	10 Gb/s	54 Mb/s	41,25 Gb/s	101,5 Gb/s
Flexibilität	sehr hoch	sehr hoch	Mod. Format	-

Tabelle 5.8 fasst die Eigenschaften in Bezug auf Performanz und Flexibilität einiger Architekturen aktueller OFDM-Veröffentlichungen zusammen. Die Werte zur Modulation beziehen sich hierbei auf die durchgeführten Hardware Experimente. In einigen Fällen, beispielsweise bei Yeh [117] wurde die erreichte Datenrate nur indirekt durch die Realisierung eines Standards angegeben. Dabei kennzeichnet ein grüner Hintergrund einen besseren Wert als das Ergebnis dieser Arbeit, ein roter Hintergrund weist auf einen schlechteren Wert hin. Gelb sind die Werte gekennzeichnet, welche sich im gleichen Rahmen bewegen.

Es ist leicht zu erkennen, dass sich die Systeme in die zwei bereits genannten Kategorien einteilen lassen:

1. Mehr Durchsatz - aber weniger Flexibilität
2. Ähnliche Flexibilität - weniger Durchsatz

Nachfolgend soll aus jeder Kategorie stellvertretend ein System genauer verglichen werden. Für Kategorie 1 zeigt Tabelle 5.9 das System aus [75] und vergleicht dieses mit der 4-SPS-Stream Implementierung des Empfänger-Prototyps aus dieser Arbeit.

Tabelle 5.9.: Der Vergleich mit einem System aus Kategorie 1, also mit höherer Datenrate und weniger Flexibilität. Zum Vergleich wurde die 4-SPS-Stream Empfängerimplementierung verwendet.

	Diese Arbeit	**Qian et al. [75]**
Jahr	2014	2010
Architektur	MPSoC	Dediziert
FPGA	XC6VHX380T	16x XC5VSX95T
Logic Cells	**382 464**	1 507 328
Samplerate	1 GSa/s	**20 GSa/s**
16-QAM Datenrate	3,09 Gb/s	**41,25 Gb/s**
Flexibilität	sehr hoch	Modulationsformat
E-B2B BER[a]	2×10^{-11}	$< 10^{-5}$
O-B2B BER[a]	**$2{,}17 \times 10^{-5}$**	$5{,}9 \times 10^{-4}$

[a] Bezogen auf Messungen mit 16-QAM

Zunächst fällt auf, das Qian et al. noch die Vorgänger Generation an FPGAs, die Virtex-5 FPGAs einsetzen. Da sich diese zwei Generationen allerdings nur gering im Aufbau ihrer Basiselemente unterscheiden, ist dies zunächst nicht weiter relevant. Qian et al. verwenden für ihren Prototyp allerdings 16 große FPGAs des Typs XC5VSX95T, der laut der Virtex-5 Produkttabelle 94 208 Logikzellen gleichgesetzt werden kann. Dies entspricht etwa der 4-fachen Menge an verfügbaren Ressourcen. Allerdings muss man bedenken, dass eine Verteilung auf 16 FPGAs auch einen erheblichen Mehraufwand und eine große Menge an ungenutzten Ressourcen bedeutet, da es selten möglich ist, einen FPGA vollständig auszulasten.

Der Vergleich der Flexibilität und der erreichten Datenrate ist eindeutig. Zwar fallen die erreichten Datenraten dieser Arbeit um ein 10-faches kleiner aus, dafür lässt sich bei Qian et al. auch nur zwischen 3 Modulationsformaten, nämlich Modulationsformaten QPSK, 8-PSK und 16-QAM, wählen. Die Möglichkeiten der

Konfiguration zur Designzeit und der Programmierung zur Laufzeit wie sie der Empfänger dieser Arbeit bietet sind um ein Vielfaches höher.

Für den Vergleich der BER wurden die EVM_M Werte aus Abbildung 5.15 mit der Hilfe von Formel 2.27 in BER Werte umgerechnet. Während für die elektrischen Back-to-Back (Electrical Back-to-Back, E-B2B) BER Werte ein genauer Vergleich schwierig ist, da Qian et al. nur angeben, dass der Wert kleiner als $10^{-}5$ ist, erreicht die Signalqualität des Empfängers dieser Arbeit eine niedrigere BER für die optischen Back-to-Back Werte.

Allerdings sind solche Vergleiche nur bedingt aussagekräftig. Zum einen verwendete der AWG in den Messungen von Qian et al. offline produzierte Daten, während der AWG der Messungen in dieser Arbeit die aufgenommenen, online produzierten Daten enthielt. Zum anderen verwendeten Qian et al. im Gegensatz zu den Messungen dieser Arbeit einen gemeinsamen Takt, was diversen Synchronisationsproblemen vorbeugt. Somit wird zwar die Signalqualität verbessert, allerdings werden hierdurch die Anforderungen an ein praktisch nutzbares Kommunikationssystem nicht wirklich erfüllt.

Ebenso hängt die BER in großem Maße von den im System befindlichen Rauschquellen ab, wie beispielsweise Analogwandlern, Modulatoren oder Verstärkern. So ist davon auszugehen, dass der in [75] erwähnte 40GS/s ADC mit 2 Kanälen eine wesentlich geringere Auflösung besitzt als die hier verwendeten ADCs von Texas Instruments.

Stellvertretend für die Kategorie 2 vergleicht Tabelle 5.10 die Eigenschaften des 4-Stream MPSoC- Empfängerprototyps dieser Arbeit mit der Espresso Plattform aus [35]. Bei Espresso handelt es sich um eine dieser Arbeit im Grundgedanken ähnlichen Architektur, eine durch mehrere HW-Beschleuniger unterstützte CPU.

Tabelle 5.10.: Der Vergleich mit einem System aus Kategorie 2, welches ebenfalls aus einer CPU und HW-Beschleunigern besteht. Es erreicht eine geringere Datenrate, bietet jedoch ähnliche Flexibilität.

	Diese Arbeit	**Harju [35]**
Jahr	2014	2006
Architektur	MPSoC	MPSoC
Realisierung	FPGA	IC
CPU-Subsystem	Leon-3 CPU	COFFEE CPU
Anzahl Beschleuniger	23	3
Samplerate	**1 GSa/s**	20-80 MSa/s
Datenrate	**3,09 Gb/s**	54 Mbit/s
Flexibilität	sehr hoch	sehr hoch

Das Ziel von Harjus Arbeit war es, ein auf den einfachen RISC Prozessor COFFEE basiertes System zu entwickeln und durch Coprozessoren soweit zu beschleunigen, dass es den Anforderungen von Wideband Codemultiplexverfahren (Wideband Code Division Multiplexing, WCDMA), denen von OFDM im Rahmen von Universal Mobile Telecommunications System (UMTS) und denen vom 802.11a Standard genügt, ohne die Energieeffizienz zu stark zu beeinträchtigen. Das System wurde hierbei in einer 13 μm Technologie implementiert und simuliert. Ein Chip wurde jedoch nicht gefertigt.

Während der erstellte Prototyp dieser Arbeit versucht, die Datenrate zu optimieren, indem alle Verarbeitungsschritte durch Beschleuniger übernommen werden, verwendet Espresso lediglich drei Beschleuniger. Für OFDM kümmert sich einer dieser Beschleuniger um die Ein- und Ausgabe, einer um die Synchronisation und der letzte übernimmt die Fast Fourier Transformation. Die Kanalschätzung und Korrektur wird weiter vom RISC-Prozessor durch ein entsprechendes Software-Programm erledigt. Dies ist ausreichend, um die geforderten Datenraten für UMTS und 802.11a zu erreichen, welche lediglich im zweistelligen Megabit-Bereich liegen und somit weit unter den angestrebten Datenraten des Systems dieser Arbeit.

Die Espresso-Architektur ist dabei ähnlich flexibel wie die Architektur dieser Arbeit. So können mit der CPU prinzipiell alle Signalverarbeitungsschritte realisiert werden, allerdings wirkt sich dies schnell auf die erreichbare Datenrate aus, da die CPU dann einen Flaschenhals bildet. Die HW-Beschleuniger sind sehr flexibel programmierbar, allerdings ist der Espresso Demodulationsprozessor, welcher FFTs berechnet, auf eine Stützpunktanzahl von maximal 1024 beschränkt. Eine Flexibilität zur Designzeit besitzt die Espresso Plattform nicht, Änderungen müssten demnach - falls überhaupt möglich - von Hand in die Hardwarebeschreibung eingepflegt werden.

Im Bereich der Kosten und des Energieverbrauchs kann die Architektur dieser Arbeit natürlich nicht mithalten. Das Ziel der Realisierung eines Systems, welches die Lücke zwischen Kategorie 1 und Kategorie 2 schließt, also sehr hohe Geschwindigkeiten ermöglicht und trotzdem die flexible Anpassung der OFDM-Parameter zur Laufzeit erlaubt, wurde jedoch erreicht.

Abschließend soll angemerkt werden, dass die erreichten Datenraten durchaus noch gesteigert werden können und es sich bei den Beschränkungen um solche der Prototypen handelt. So kann zukünftig, unter der Verwendung von FPGAs der nächsten Generation, die mehr Ressourcen bieten oder die einen höheren Systemtakt erlauben, der Durchsatz entsprechend erhöht werden.

Auch die Realisierung durch einen ASIC ist potentiell möglich. Hier könnte man die Taktrate des Systems deutlich steigern und es würde einen wesentlich höheren Durchsatz erzielen. Einerseits müsste man jedoch auf eine effiziente Umset-

zung derjenigen Elemente achten, welche derzeit durch BRAMs realisiert sind, andererseits würde man als ASIC natürlich jegliche Flexibilität zur Designzeit verlieren.

6. Entwurf eines Hochgeschwindigkeits-Digital Down Converters

Dieses Kapitel beschreibt die Realisierung eines Digital Down Converters (DDC), welcher für das in Kapitel 4.1 beschriebene Anwendungsszenario zur Vorverarbeitung der Signale im Empfänger benötigt wird. Ausgehend von der allgemeinen Funktionsweise und den benötigten Eigenschaften wird zunächst die Architektur ermittelt, welche die gesetzten Anforderungen erfüllt. Danach wird die Implementierung im Detail besprochen, um im letzten Kapitel die Experimente und Ergebnisse der Implementierung vorzustellen

6.1. Digital Down Converter zur OFDM Vorverarbeitung

Eine Digital Down Conversion ist häufig in Kommunikationssystemen zu finden, weshalb sie bereits vielfach analysiert und optimiert wurde. Allerdings besitzt das digitale Verarbeitungssystem im Regelfall eine relativ niedrige Samplerate, die Abtastfrequenz ist normalerweise geringer als die Taktfrequenz des Systems.

Die Forderung nach Sampleraten von mehreren Gigasamples pro Sekunde, die für den Einsatz in Hochgeschwindigkeits-OFDM Systemen erforderlich sind, verhindert jedoch den Einsatz von Standard-Realisierungen. Liegt die Samplingfrequenz bei einem Vielfachen des Systemtaktes, müssen entsprechend viele Samples pro Takt zeitgleich und unter Verwendung des Pipelining Prinzipes verarbeitet werden können.

6.2. Aufbau und Funktion eines Digital Down Converters

Der allgemeine Aufbau einer Digital Down Conversion lässt sich in zwei bis drei Stufen unterteilen und ist in Abbildung 6.1 veranschaulicht. In der ersten Stufe muss das zu untersuchende Signal in das Basisband gemischt werden, was hier durch eine komplexe Multiplikation mit dem negierten Träger des Signals geschieht. Dieser Träger kann als digitales Signal mit Hilfe eines *numerisch gesteuerten Oszillators* (Numerically Controlled Oscillator, NCO) erzeugt werden.

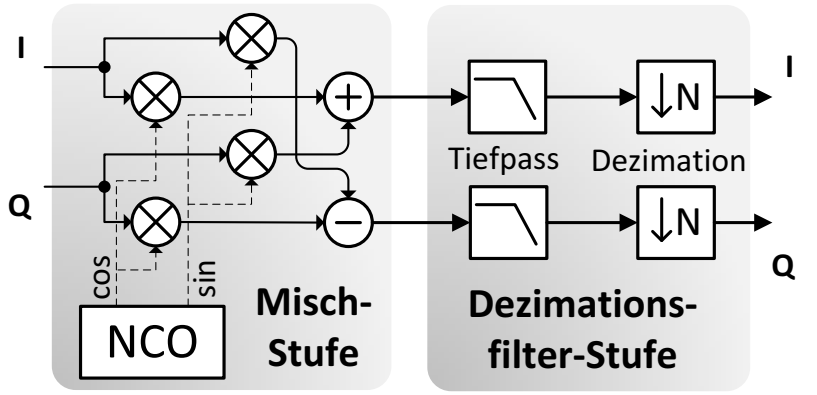


Abbildung 6.1.: Im Allgemeinen besteht eine Digital Down Conversion aus einer komplexen Multiplikation mit der negierten Trägerfrequenz sowie der anschließenden Tiefpassfilterung und Verringerung der Samplerate um einen Faktor N.

Im Bild 6.1 wird eine allgemeingültige Realisierung dargestellt. Je nachdem ob es sich bei dem verwendeten Eingangssignal um ein IQ-Signal handelt oder um ein reelles Signal, lässt sich die Mischstufe auf zwei reelle Multiplikationen vereinfachen. Kann bei einem reellen Eingangssignal einfach der Q-Eingangspfad entfernt werden, genügt bei einem IQ-Eingangssignal die Verwendung eines reellen Trägersignals. Unerwünschte Spiegelspektren werden durch nachfolgende Stufen entfernt.

Nach der Mischstufe wird sowohl für den I-Pfad als auch für den Q-Pfad das Signal in der zweiten Stufe durch einen Tiefpass gefiltert, um es auf die gewünschte Bandbreite zu beschränken und so auf ein Reduzieren der Samplingfrequenz in der dritten Stufe vorzubereiten. Ein geschicktes Integrieren der Sampleratendezimierung in die Filter kann allerdings eine effizientere Realisierung dieser zwei Funktionen ermöglichen. Daher werden Stufe zwei und drei oft gemeinsam durch einen Dezimationsfilter realisiert.

Dieser (Dezimations-)Filter des Systems erfordert typischerweise große Aufmerksamkeit, da die Anforderungen an das Filter zu einem großen Teil die Komplexität und den Ressourcenverbrauch der gesamten DDC bestimmen. Dabei ist es wichtig, dass linearphasige Filter verwendet werden, um eine spätere Korrektur der Werte zu erleichtern.

Für die meisten Standard DDC-Realisierungen mit einer hohen Dezimation wird eine Kombination aus Cascaded-Integrator-Comb-Filter (CIC-Filter) und einem

oder mehreren nachfolgenden Finite-Impulse-Response-Filter (FIR-Filter) verwendet, vergleiche den GC4016 Multi-Standard Quad DDC Chip von Texas Instruments [41] und [90]. CIC-Filter ermöglichen die effiziente Realisierung hoher Dezimationsraten. Nachteile im Passbandverhalten werden durch spezielle FIR-Realisierungen ausgeglichen.

Für niedrige Dezimationsraten, kleiner gleich sechzehn, stellen FIR-Filter und kaskadierte Goodman und Carey Halbband-FIR-Filter [30] eine Alternative dar. Da diese Filter keine Rückführung des Ausgangs besitzen, lassen sich mit Hilfe einer Polyphase-Realisierung wesentlich einfacher mehrere dieser Filter in einer parallelen Struktur realisieren. Sie können zusammen somit mehr als ein Sample pro Takt verarbeiten. Für die konkreten Anforderungen der Experimente dieser Arbeit stellen kaskadierte Goodman und Carey Halbband-FIR-Filter die bessere Wahl dar. Weitere Details hierzu, sowie der Vergleich zu einer entsprechenden Polyphase-FIR-Filter Version, wurden bereits in [Men12] und [MMD+12] veröffentlicht.

6.2.1. Anforderungen im Rahmen der Arbeit

Ziel ist es, mit Hilfe eines in Kapitel 5.1.1 beschriebenen ADC30 Wandlers von Micram ein Signal mit einer sehr hohen Bandbreite abzutasten, daraus ein beliebiges Signalband auszuschneiden und dieses Band anschließend auf eine für den Empfänger realisierbare Samplerate von ein bis zwei Gigasample zu bringen. Hierzu wird eine Reduktion der Samplerate um den Faktor 16 angestrebt. Betreibt man den ADC dann lediglich mit 16 Gigasample pro Sekunde, erhält man eine Ausgangssamplerate von 1 GSa/s.

Dieses Verhältnis lässt sich bei Bedarf linear bis zu einer ADC Samplerate von 30 GSa/s und einer Samplerate des DDC-Ausgangssignals von 1,875 GSa/s steigern. Bei einer angestrebten Samplerate von einem Gigasample für die Sender- und Empfängerimplementierung nimmt die DDC demnach bei 125 MHz Systemtakt und 16 GSa/s Samplerate 128 Eingangssample pro Takt entgegen und dezimiert diese zu 8 Werte pro Takt.

6.3. Implementierung

Die Struktur des entwickelten Hochgeschwindigkeits-Digital Down Converters ist in Abbildung 6.2 dargestellt. Zunächst wird durch den *Input Scaler* Block die Zahlendarstellung angepasst. Während der darauf folgende NCO der Mischstufe aus 128 kooperierenden NCOs besteht, welche bereits im OFDM Empfänger zum Einsatz kommen, stellt der Dezimationsfilter eine Kaskadierung von zwei

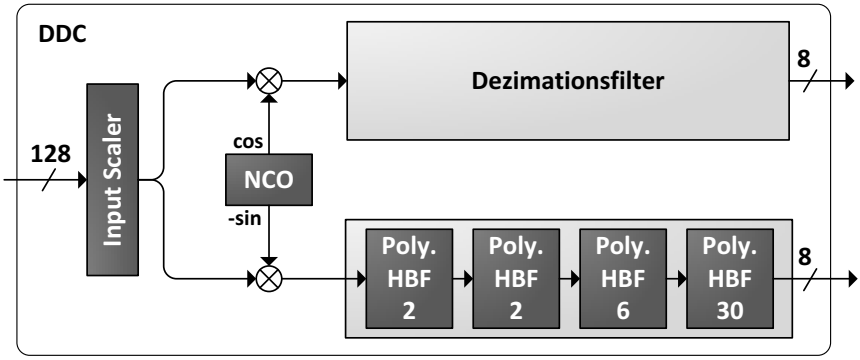


Abbildung 6.2.: Der realisierte Hochgeschwindigkeits-Digital Down Converter dezimiert 128 Samples mit 6 Bit Auflösung pro Takt zu 8 Samples mit 16 Bit Auflösung pro Takt. Dabei kommen als Dezimationsfilter mehrere kaskadierte Halbbandfilter zum Einsatz.

Halbbandfiltern zweiter Ordnung, eines Halbbandfilters sechster Ordnung und zuletzt eines Halbbandfilters dreißigster Ordnung dar. Um bei den anvisierten Sampleraten eine Implementierung zu erreichen, welche sich mit den Möglichkeiten heutiger FPGAs realisieren lässt, wurden alle Komponenten von Hand aufgebaut und optimiert. Für einen optimalen Aufbau der Halbbandfilter sollte man diese so beschreiben, dass die Xilinx Synthesewerkzeuge die Möglichkeiten der DSP48E1 Elemente vollständig nutzen können.

Zu Beginn einer Filterstufe steht immer ein Verzögerungsblock, welcher zur Entspannung der Timing-Anforderungen die Eingangsdaten registriert sowie durch zusätzliche Register dafür sorgt, dass genügend Eingangsdaten für alle parallelen Filter vorhanden sind. In Abbildung 6.3 ist dieser Block auf der linken Seite veranschaulicht.

6.3.1. Aufbau des Dezimationsfilters durch Halbbandfilter

Hinter dem Verzögerungsblock kommen parallel mehrere Polyphase-Halband-Filter zum Einsatz. Da die Grenzfrequenz von Halbbandfiltern bei $\omega_g = \pi/2$, also in der Mitte des Spektralbands liegt, lässt sich mit jeder Halbbandfilterstufe eine Verringerung der Samplerate um den Faktor 2 durchführen. Dementsprechend kann die Realisierung jedes zweiten Filters eingespart werden, da dessen Ergebnis im Sinne der Sampleratenreduzierung nicht weiter verwendet wird. Abbildung 6.3 veranschaulicht dieses Konzept.

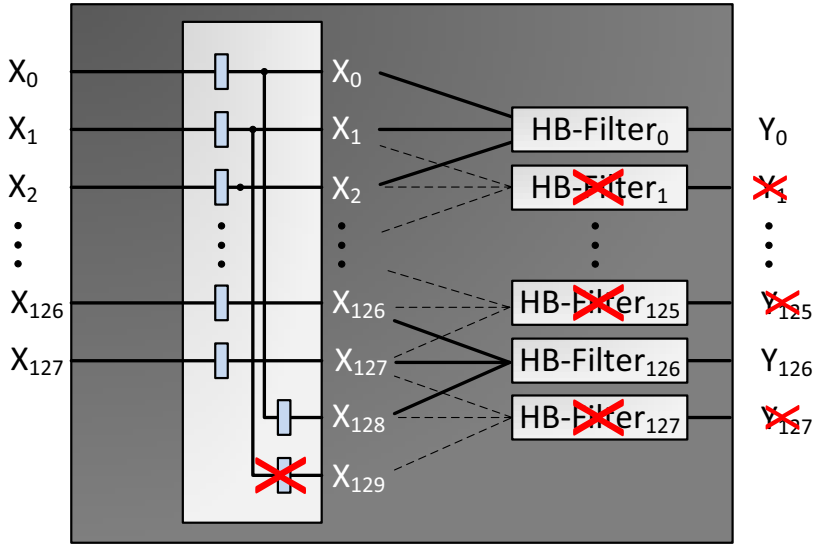


Abbildung 6.3.: Nach dem Verzögerungsblock kann durch die Sampleratenreduktion um den Faktor 2 auf die Realisierung jedes zweiten Polyphase-Halband-Filters verzichtet werden.

Die Koeffizienten reeller Halbbandfilter sind symmetrisch, wobei bis auf den mittleren alle Koeffizienten mit geradem Index null sind. Der mittlere Koeffizient lässt sich allerdings durch Skalierung zu 1 festlegen, wodurch man für Halbbandfilter ungefähr ein Viertel der üblichen Multiplikationen benötigt.

Ein Vorteil des Aufbaus mit Hilfe von kaskadierten Halbbandfiltern besteht darin, dass das Passband der ersten Filterstufen viel größer ist als der entsprechende Durchlassbereich der gesamten Filterbank. Die vordersten Filter können somit sehr einfach gehalten werden, da hier der Abstand zwischen Sampling-Frequenz und Nutzsignal hoch ist. Sie dürfen sehr breite Übergangsbereiche aufweisen. Die Dämpfung dieser Bereiche wird im Gesamtfilter von nachfolgenden Filtern übernommen. So ist hier bereits mit einstelligen Filterordnungen eine hohe Sperrdämpfung möglich.

Entsprechend werden mit steigendem Verhältnis von Passband zu Samplingfrequenz Filter höherer Ordnung benötigt, damit die gewünschte Aliasingunterdrückung zu erhalten. Goodman und Carey [30] haben eine Tabelle einfach zu implementierender Halbbandfilter mit ansteigender Länge aufgestellt sowie ein

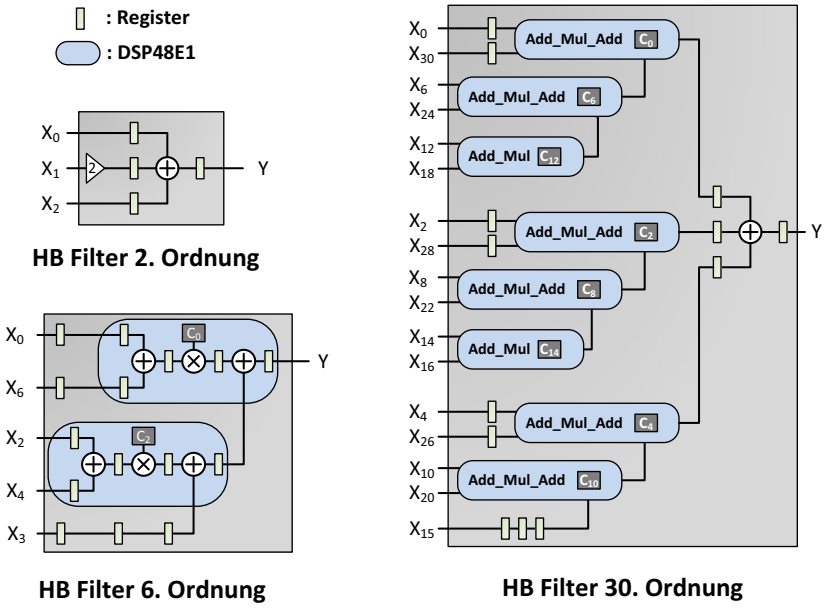


Abbildung 6.4.: Die einzelnen Halbbandfiltertypen des DDC-Design: a) 2. Halbbandfilter Ordnung b) Halbbandfilter 6. Ordnung c) Halbbandfilter 30. Ordnung

Verfahren, um zu einer gewünschten Reduktionsrate und Dämpfung die jeweils passenden Filter für jede Stufe auszuwählen. Nach diesem Verfahren wurden die Ordnungen der Filterkaskade gewählt.

Abbildung 6.4 zeigt, wie die zum Einsatz kommenden Halbbandfilter optimal mit Hilfe von DSP48E1 Slices aufgebaut werden können. Da die Halbbandfilter zweiter Ordnung ganz ohne Multiplikation auskommen, werden für ihre Realisierung auch keine DSP48E1 Primitive benötigt, siehe Abbildung 6.4 a). Trotzdem kann man ihre Implementierung optimieren, indem man durch eine geeignete Beschreibung dafür sorgt, dass die Synthese Tools sogenannte *Ternary Adders* verwenden. Diese können direkt drei Summanden addieren benötigen allerdings Dank der LUTs mit sechs Eingängen den gleichem Ressourcenaufwand wie eine Addition zweier Summanden, siehe [101]. Dies ist insbesondere für große Addiererbäume, wie sie in digitalen Filtern vorkommen, von Vorteil.

Die Realisierung eines HB-Filters 6. Ordnung ist in Abbildung 6.4 b) zu sehen. Durch die Symmetrie der Koeffizienten können die Pre-Adder der DSP48E1 Primitive genutzt werden, um die entsprechenden Eingangsdaten zu addieren und entsprechend des Distributivgesetzes nur das Ergebnis dieser Addition mit dem Koeffizienten zu multiplizieren. Die Post-Adder der DSP48E1 Slices werden verwendet, um die Einzelergebnisse zu addieren. Dadurch kann, abgesehen von ein paar Registern, welche im Bild durch hellgrüne Rechtecke dargestellt sind, ein einzelnes HB-Filter sechster Ordnung bereits mittels zwei DSP48E1 Slices (im Bild blau hervorgehoben) aufgebaut werden.

Tabelle 6.1.: Ressourcenverbrauch aufgeschlüsselt auf einzelne Komponenten des DDC.

Komponente	**LUTs**		**Flip Flops**		**BRAMs**		**DSP48**	
Vollständiges DDC Design[a]	40490	17%	64510	13%	83	11%	193	22%
Analogschnittstelle[b]	14922	37%	16870	26%	54	65%	1	1%
Chipscope[b]	2364	6%	4394	7%	29	35%	0	0%
DDC[b]	22953	57%	42476	66%	0	0%	192	99%
Mischstufe[c]	7487	33%	7978	19%	0	0%	0	0%
Erste HB-Filterstufe[c]	1270	6%	3399	8%	0	0%	0	0%
$\rightarrow$ HB-Filter 2. Ordnung[c]	19	0%	36	0%	0	0%	0	0%
Zweite HB-Filterstufe[c]	820	4%	2091	5%	0	0%	0	0%
$\rightarrow$ HB-Filter 2. Ordnung[c]	20	0%	44	0%	0	0%	0	0%
Dritte HB-Filterstufe[c]	233	1%	1392	3%	0	0%	32	17%
$\rightarrow$ HB-Filter 6. Ordnung[c]	5	0%	60	0%	0	0%	2	1%
Vierte HB-Filterstufe[c]	1615	7%	5994	14%	0	0%	64	32%
$\rightarrow$ HB-Filter 30. Ordnung[c]	181	1%	659	2%	0	0%	8	4%

[a] Prozentangaben beziehen sich auf verfügbare Ressourcen eines XC6VHX380T FPGAs
[b] Prozentangaben beziehen sich auf vollständiges FPGA Design
[c] Prozentangaben beziehen sich lediglich auf den Digital Down Converter

Der komplexeste Einzelfilter, der HB-Filter 30. Ordnung, ist in Abbildung 6.4 c) zu sehen. Wie bei einem Filter sechster Ordnung kommen DSP48E1 Primitive zum Einsatz, welche mit den Pre-Addern die entsprechenden Eingangsdaten addieren, die Multiplikation der Koeffizienten realisieren sowie mit Hilfe der Post-Adder Teile des benötigten Addiererbaums implementieren. Diese Module werden im Bild *Add_Mul_Add* genannt. Des Weiteren finden sich *Add_Mul* Module, welche die letzte dieser Stufen, den Post-Adder und das entsprechende Register, nicht verwenden. Zudem kommt auch hier ein Ternary Adder zum Einsatz.

Der Ressourcenverbrauch einer DDC ist in Tabelle 6.1 aufgeschlüsselt dargestellt. Angemerkt sei, dass sich Angaben zu Filterstufen und zur Mischstufe nur auf eine

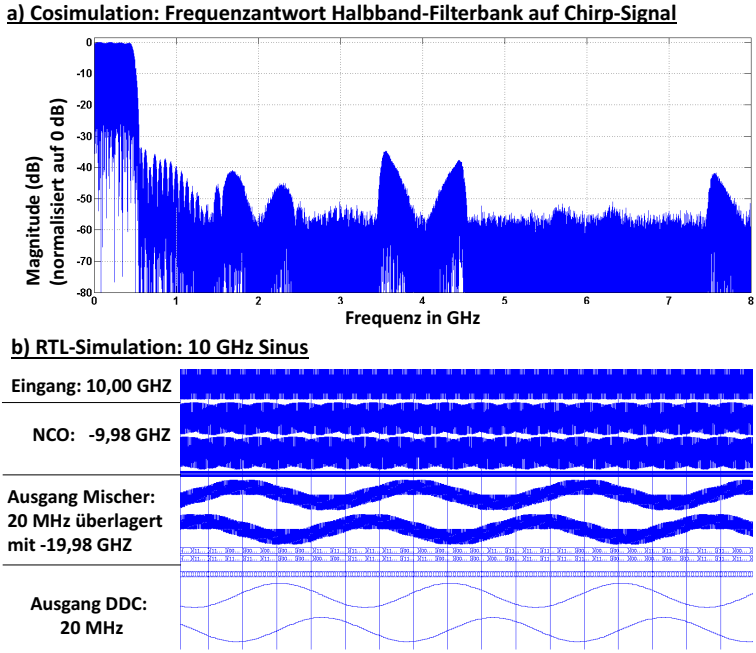


Abbildung 6.5.: a) Ergebnis einer Matlab-Modelsim-Cosimulation welches die Frequenzantwort der Filterkaskade auf ein Chirpsignal zeigt. b) Modelsim RTL-Simulation in welcher ein (reelles) 10 GHz Signal absichtlich mit einem 20 MHz Versatz ins Basisband gemischt wird.

Realisierung beziehen, in einem DDC-Design allerdings immer zwei, je eine pro IQ-Pfad, benötigt werden. Jedoch gilt es zu bedenken, dass die internen Bitbreiten innerhalb des DDC ansteigen, die Anzahl der Einzelfilter pro Filterstufe allerdings wegen der Sampleratenreduktion pro Stufe abnimmt. Für die Mischstufe am Eingang wurde der Einsatz von DSP48E1 Elementen absichtlich verhindert, da hier die Eingangsdaten nur 6-Bit breit sind und sich die Mischstufe mit Hilfe von genügend Pipeline Registern auch aus CLBs effizient und leistungsfähig bauen lässt. Die wertvollen DSP48E1 Primitive können so für andere Aufgaben eingespart werden.

Die Simulationsergebnisse erster einfacher Tests des Digital Down Converter Designs sind in Abbildung 6.5 zu sehen. Teilbild a) zeigt die Frequenzantwort auf

ein Chirpsignal, welches im Rahmen einer Cosimulation in Matlab erzeugt, mittels Modelsim durch die Dezimationsfilterstufe des DDC verarbeitet und letztlich durch Matlab ausgewertet wurde. Wie erwartet schließt das Filter bei 500 MHz. Teil b) zeigt den Test des ganzen DDC-Designs in einer puren RTL-Simulation. Als Eingangssignal dient ein 10 GHz Sinussignal, der Quadraturmischer des DDC wird auf -9,98 GHz eingestellt, so dass sich eine Überlagerung von 20 MHz und -19,98 GHz ergibt. Nachdem in der Dezimationsfilterkaskade der negative 19,98 GHz Anteil entfernt und die Samplerate um den Faktor 8 reduziert wurde, findet sich am Ausgang des DDC ein 20 MHz Signal.

6.4. Aufbau des DDC-Experiments und Auswertung der Messergebnisse

Für komplexere Experimente wurde das DDC Design mit Hilfe des Hitech Global Entwicklungsboards realisiert. Im Testaufbau kommt wieder ein AWG als Sender zum Einsatz, das DDC-Design nutzt einen Vega-ADC30 Analog/Digital-Wandler von Micram, um ein Eingangssignal mit einer Abtastrate von 16 GSa/s zu digitalisieren, vergleiche Abbildung 6.6.

Anschließend wird das enthaltene OFDM-Signal durch die Mischstufe des DDC-Designs ins Basisband gemischt. Am Ende der Verarbeitungskette dieses Experimentes steht das gefilterte und in der Samplerate reduzierte OFDM-Signal im Basisband. Wie in Abbildung 6.6 in der oberen Hälfte dargestellt, ermöglicht der on-Chip Logikanalysator Chipscope das Auslesen der Daten sowohl vor als auch hinter der DDC-Komponente.

Die Ergebnisse zur Bewertung des DDC-Designs sind in Abbildung 6.7 dargestellt. Ganz links zeigt das Teilbild A das Konstellationsdiagramm des verwendeten Sendesignals. Hierfür wird sowohl die DDC-Funktionalität als auch die Demodulation in Matlab ausgeführt. Dabei wurde nicht versucht, die Filterstruktur der Hardware nachzustellen, sondern es kamen entsprechend optimierte Matlab-Funktionen zum Einsatz, um ein optimales Ergebnis zu erzielen. Der sich ergebende EVM_M Wert von 5,63% kann daher als Referenz gewertet werden.

Teilbild B zeigt das Konstellationsdiagramm, welches man erhält, wenn das Sendesignal direkt digital dem DDC-Design zur Verfügung gestellt und nicht durch den AWG und den Micram ADC beeinträchtigt wird. Die Demodulation der Ausgabedaten des Converters wurde offline in Matlab durchgeführt. Der EVM_M Wert von 5,98% kann somit zur Bewertung des DDC-Design verwendet werden. Er belegt nicht nur, dass die realisierte DDC funktioniert. Der Vergleich mit der Matlab Referenzimplementierung zeigt ebenso, dass die Degradierung des Signals,

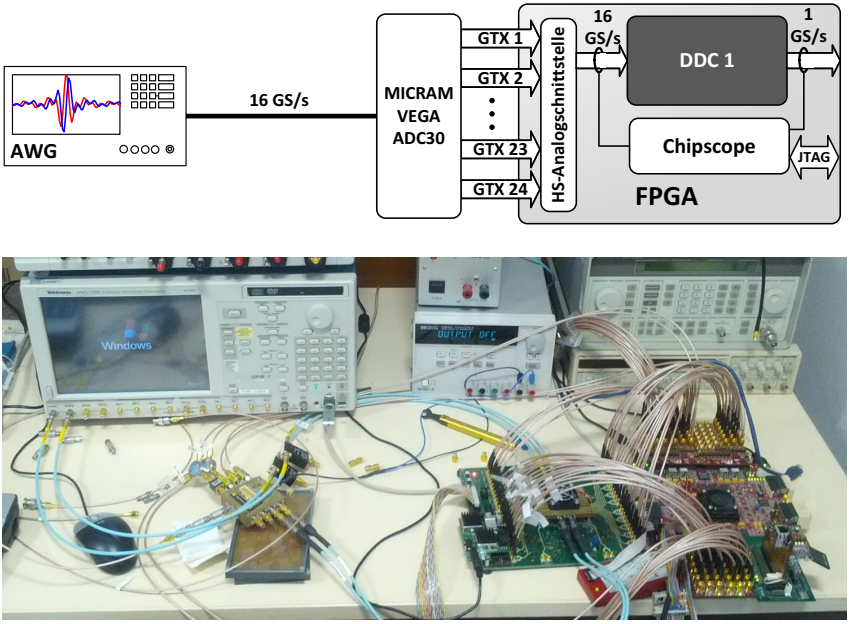


Abbildung 6.6.: Das DDC Testsystem erlaubt die Analyse des Signals vor sowie hinter dem Digital Down Converter durch den on-Chip Logikanalysator Chipscope. In der oberen Hälfte ist der Aufbau als Blockdiagramm dargestellt, unten sieht man eine Photographie des Aufbaus.

welche durch die Einschränkungen wie Bitbreiten etc. zustande kommen, sich in einem tolerierbaren Rahmen bewegen.

Das Konstellationsdiagramm aus Teilbild C zeigt die Ergebnisse der Messungen mit analoger Übertragung. Man erhält ein solches Diagramm, wenn man das Empfangssignal mit Hilfe von Chipscope am Eingang des Digital Down Converters aufzeichnet und dessen Funktion als auch die Demodulation entsprechend Teilbild A durch Matlab ausführt. Der EVM_M Wert von 25% stellt im Vergleich zum EVM_M Wert aus Messung A die Degradierung des Signals dar, welche aus der Verwendung des AWG und aus Verwendung des Hochgeschwindigkeits-ADC resultiert.

Das letzte Teilbild (D) zeigt das Ergebnis der Matlab Demodulation des Signals, welches sich mit Hilfe von Chipscope hinter der Digital Down Conversion auf-

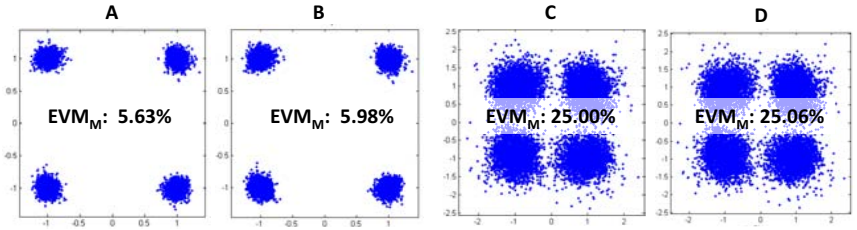


Abbildung 6.7.: Auswertung des DDC. A: Sendesignal B: Degradierung durch Echtzeit-DDC C: Degradierung durch die analoge Übertragung D: Ergebnisse der analogen Übertragung inklusive Echtzeit-DDC

zeichnen lässt. Auch hier wird der erzielte EVM_M Wert von 25,06% nur geringfügig schlechter als der Wert der Vergleichsmessung C, in welche die Matlab Referenzimplementierung der DDC verwendet wurde.

Die Experimente zeigen, dass die realisierte DDC im Vergleich der Matlab-Referenz nur geringfügig schlechtere Ergebnisse liefert. Vergleicht man das Ergebnis aus Teilbild A mit dem aus B, sowie das Ergebnis aus Teilbild C mit dem aus D, erkennt man, dass die Degradierung der EVM_M durch die Hardware-Realisierung im Vergleich zu den Simulationen äußerst gering ausfällt.

Zwar wurden in den Experimenten mit einer QPSK Modulierung der Subträger EVM_M Werte erreicht, welche im akzeptablen Bereich liegen (siehe Kaptel 2.1.7), dennoch ist die Verschlechterung des Signals durch den Einsatz des MICRAM Wandlers enorm. Es muss jedoch angemerkt werden, dass dieser Analog/Digital-Wandler eine aufwendige Kalibrierung benötigt. Diese Kalibrierungsroutinen wurden vom Hersteller des Wandlers zum Zeitpunkt der Messungen noch nicht veröffentlicht. Eine selbst implementierte Kalibrierung war aufgrund mangelnder Informationen und Möglichkeiten nur eingeschränkt möglich. Korrekt kalibriert, sind erheblich bessere Ergebnisse zu erwarten.

7. Organisation und Entwurfsfluss des HW/SW Co-Design Frameworks

7.1. Aufgabe des Frameworks

Ein Ziel dieser Arbeit ist es, auch Systementwicklern den Aufbau eines Hochgeschwindigkeits-OFDM Systems zu ermöglichen, welche wenig Expertise im Entwurf digitaler Schaltungen und eingebetteter Systeme besitzen. Daher wurde während der Entwurfs- und Implementierungsphase stets auf ein modulares Design geachtet, sowie auf eine zentrale Verwaltung von Designoptionen durch VHDL-Generics Wert gelegt. Hierdurch ermöglicht es das Framework nicht nur, die einzelnen Komponenten individuell wiederzuverwenden, sondern auch eine Vielzahl unterschiedlicher Varianten zu generieren.

Das Framework erleichtert dabei nicht nur zur Designzeit großen Einfluss auf die Anzahl und auf die Typen von Hardware-Beschleunigern zu nehmen, es unterstützt auch das Ändern der OFDM Parameter zur Laufzeit durch Software. Gerade durch die Flexibilität zur Designzeit werden daher die verschiedensten Ausprägungen eines HW/SW Co-Designs unterstützt.

Das Framework gliedert sich wie in Abbildung 7.1 dargestellt in drei Bereiche. Dabei bilden die einzelnen Komponenten aus Kapitel 4 und 5 den ersten Bereich, im folgenden auch als Hardware-Bibliothek bezeichnet. Der zweite Bereich ist durch den System Generator zur automatisierten Erstellung von OFDM-Systemen gegeben. Diese zwei Hardware Bereiche werden durch den dritten Bereich, der Software-Bibliothek zur einfachen Programmierung des Gesamtsystems, vervollständigt.

7.2. Der System Generator

Der System Generator ist im Grunde ein sehr großes und flexibles Transceiver-Design, welches aus den Prototypen der OFDM-Experimente von Kapitel 5 erstellt und um eine grafische Benutzerschnittstelle zur Erzeugung einer Konfiguration erweitert wurde. Mit seiner Hilfe ist es möglich, eine Vielzahl von Systemvarianten zu generieren, ohne dafür tiefgehende Kenntnisse über die Architektur zu be-

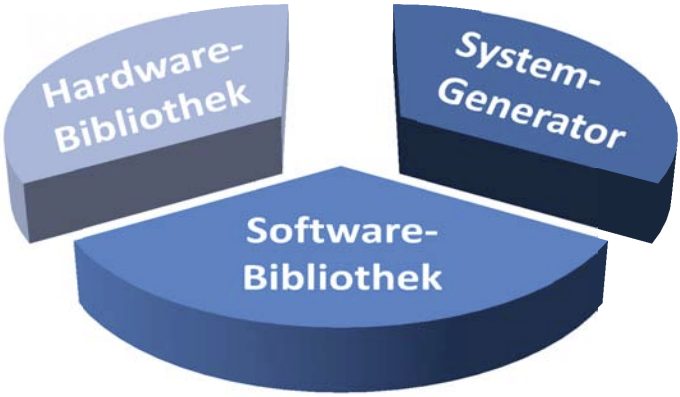


Abbildung 7.1.: Der Aufbau des Frameworks ist in drei Bereiche organisiert. Die HW-Bibliothek beinhaltet die erstellten Einzelkomponenten, der System-Generator ermöglicht die Erstellung einer Vielzahl von Systemvarianten und die SW-Bibliothek stellt vorgefertigte Treiber- und Testfunktionen zur Verfügung.

nötigen. Zwar wird bisher lediglich die verwendete Prototyping Hardware voll unterstützt, das Konzept ist aber ebenso für andere FPGA Plattformen und für andere Analogwandler gültig.

Da der System Generator auf die Hardware-Bibliothek zugreift und auch eine Systeminformationsdatei für die Softwarebibliothek erstellt, funktioniert dieser nur im Rahmen des Frameworks.

7.2.1. Konfigurationsdatei des System Generators

Das System Generator Design macht einen exzessiven Gebrauch von *VHDL-Generate-Konstrukten*, welche über *VHDL-Generics* gesteuert werden. Auf diese Weise werden sowohl die Parameter der einzelnen Komponenten des Systems, als auch die Anzahl dieser Komponenten durch diese VHDL-Generics verwaltet. Die Generics selbst befinden sich dabei zentralisiert in einer einfachen Konfigurationsdatei. Damit die Erstellung bzw. die Modifikation dieser Datei nicht von Hand vorgenommen werden muss, wurde eine grafische Eingabe (Graphical user interface, GUI) entwickelt. Zusammen mit dem HDL-Design stellt diese GUI den System Generator dar.

Durch den System Generator ist es möglich, Sender-, Empfänger-, sowie Transceiversysteme zu erstellen und gleichzeitig einen angepassten Kompromiss aus maximalem Durchsatz, Flexibilität und Ressourcenverbrauch zu bestimmen. Tabelle 7.1 listet eine Auswahl wichtiger Designparameter auf und beschreibt, auf welche Eigenschaften sich diese Parameter hauptsächlich auswirken und durch welche Ressourcen sie beschränkt sind.

Tabelle 7.1.: Architekturoptionen zur Designzeit und ihr Einfluss auf das System.

Funktionalität	**Einfluss**	**Beschränkung**
Anzahl an Beschleunigern	Durchsatz	Allg. Ressourcen
Anzahl an Speicher	Durchsatz	BRAM
Anzahl an SPS-Streams	Durchsatz	Allg. Ressourcen
Modus des Connectors	Flexibilität	Allg. Ressourcen
Modus des (i)FFT Beschleuniger	Durchsatz	DSP48E1
Max. Subträgeranzahl	Flexibilität	Allg. Ressourcen
Größe der AHB Speicher	Flexibilität	BRAM
Größe des Pilottonspeichers	Flexibilität	BRAM
Max. Anzahl Symbole pro Frame	Flexibilität	BRAM
Max. Anzahl Pilottöne pro Sym.	Flexibilität	BRAM
Max. Anzahl an EQ-Symbolen	Flexibilität	BRAM
Größe des Präambelspeichers	Flexibilität	BRAM
DAC Auflösung	-	-
ADC Auflösung	-	-

Die wichtigsten Parameter, welche die Leistungsfähigkeit und den Ressourcenverbrauch dominieren, sind die Anzahl der Beschleuniger, der Speicher sowie der SPS-Streams. Durch die maximale OFDM Trägeranzahl und die Art der Verbindungsstruktur (Punkt-zu-Punkt oder Matrix) können allgemeine Ressourcen gespart werden. Mit der Festlegung der Parameter zur maximalen Größe der Präambel, der Anzahl der Piloten und der Tiefe der Speicher kann stark Einfluss auf die benötigte Anzahl an BRAM-Blöcken genommen werden. Die Auflösung der Analogwandler hat wenig Auswirkung auf den Rest des Systems, da das Anpassen der internen 16-Bit Darstellung erst am Ende des Sender beziehungsweise bereits am Anfang des Empfängers geschieht.

7.2.2. Entwurfsfluss des Frameworks

Der Entwurfsfluss zur Erstellung von OFDM-Signalverarbeitungssystemen mittels des System Generators ist in das Entwicklungswerkzeug *ISE Design* des FPGA Herstellers Xilinx eingebunden und nutzt somit ein entsprechendes ISE-Projekt.

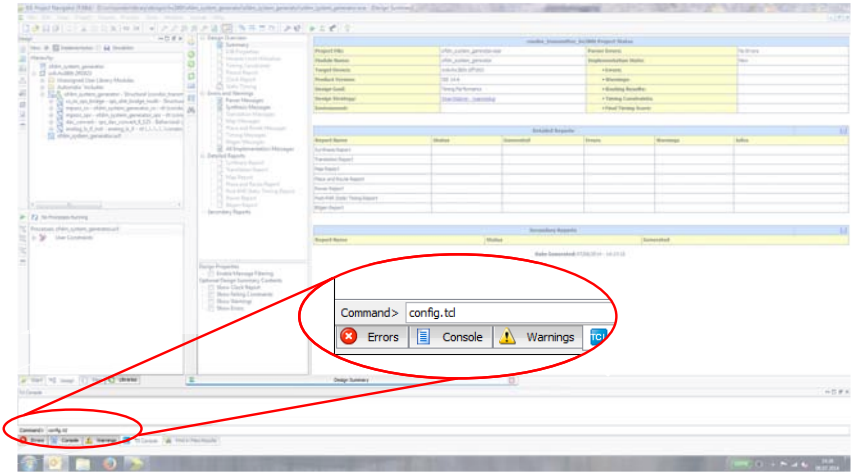


Abbildung 7.2.: Die grafische Eingabe für die Konfiguration kann über das Kommando „*config.tcl*“ aufgerufen werden.

Wird die zugehörige Projektdatei mit dem Entwicklungswerkzeug geöffnet, kann ein Design durch zwei einfache Schritte generiert werden:

1. Erzeugen der Konfiguration
2. Starten der Erstellung des Bitstroms

Um die grafische Benutzereingabe zur Erstellung einer Konfigurationsdatei aufzurufen, muss in die Konsole am unteren Rand des Entwicklungswerkzeuges das Kommando „*config.tcl*“ eingegeben werden, vergleiche Abbildung 7.2.

Die GUI selbst ist in Abbildung 7.3 dargestellt. Sie wurde mit dem Werkzeug *Graphical User Interface Development Environment* (Guide) von Matlab erstellt und ist als eigenständig lauffähiges Programm kompiliert. Sie ist in englischer Sprache verfasst und in drei Bereiche unterteilt.

Der obere Bereich umfasst die Optionen welche das Control System betreffen. Neben der Taktfrequenz, der Baudrate für die seriellen Verbindung, den Ethterneteinstellungen und den Einstellungen des Leon-3 Cache kann bestimmt werden, welche Komponenten des CS im Design realisiert werden sollen. Für alle Komponenten kann der Adressbereich und die IRQ-ID gesetzt werden. Hierfür lässt sich jedoch auch durch den Knopf in der Mitte unten automatisch ein Vorschlag generieren.

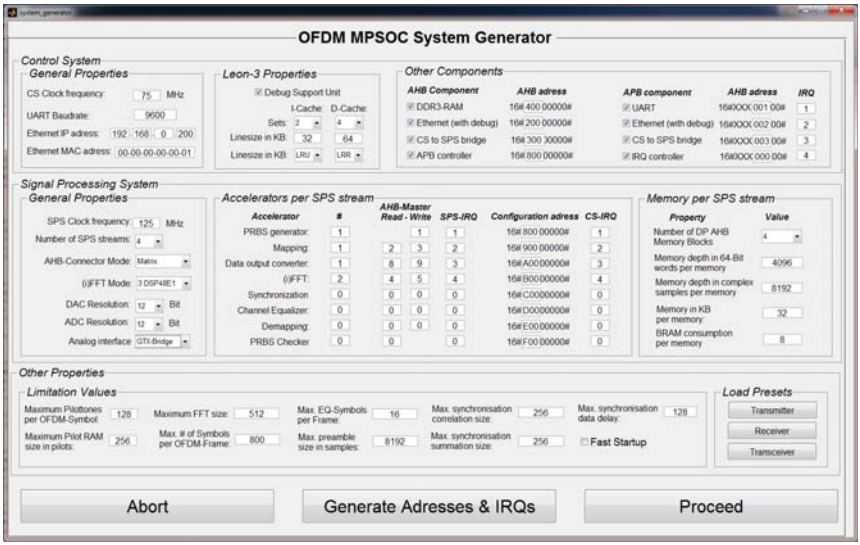


Abbildung 7.3.: Die grafischen Eingabe zur automatisierten Generierung einer gültigen Konfiguration.

Im zweiten Bereich sind die Optionen für das SPS zusammengefasst. Die wichtigsten sind durch die Taktfrequenz und die Anzahl an Streams sowie die Anzahl verschiedener Beschleuniger und AHB Speicher pro Stream gegeben. Auch die Art von analoger Schnittstelle lässt sich hier bestimmen. Zur Auswahl stehen die verwendeten Wandler dieser Arbeit direkt, oder per GTX-Brücke angebunden. So bewirkt die Option *Native*, dass keine Analogschnittstelle integriert wird. So können Designs erstellt werden, die leicht per Hand durch eine beliebige Schnittstelle ergänzt werden können.

Die Adressen der AHB Speicher werden aufsteigend vergeben, der erste Speicher hat Adresse 0x00000000, der zweite Adresse 0x00100000 und der n-te hat Adresse $(n-1) \cdot 0x00100000$. Die Adressen der Konfigurationsspeicher sowie die IRQ-IDs können, ähnlich zum CS, beliebig editiert werden. Außerdem können den Beschleunigern die Schnittstellen des AHB Connectors zugewiesen werden. All dies kann jedoch, wie schon beim CS, auch wieder über den Knopf in der Mitte unten automatisiert geschehen. Sind mehr als ein Beschleuniger eines Typs pro SPS-Stream vorhanden, belegen die zusätzlichen Beschleuniger die nächsten Schnittstellen und die nächsthöhere Adresse des Speichers.

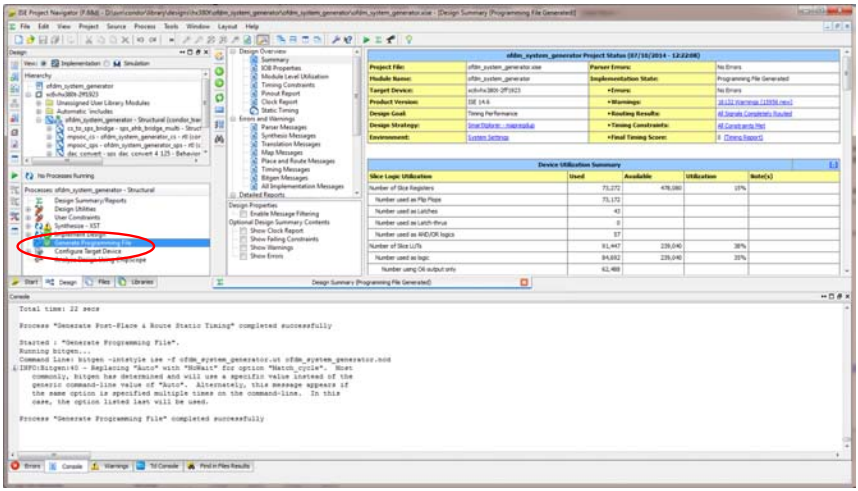


Abbildung 7.4.: Nachdem die Konfiguration mit Hilfe der GUI erstellt wurde, kann mittels des Knopfes *Generate Programming File* ein Bitstrom für den FPGA erzeugt werden.

Als Beispiel für den Umgang mit mehreren Beschleunigern werden die Einstellungen für zwei (i)FFT Beschleuniger, wie in Bild 7.3 dargestellt, näher betrachtet. Für sie ist die Leseschnittstelle mit 4, die Schreibschnittstelle mit 5 und die Konfigurationsspeicheradresse mit 0xB0000000 spezifiziert. Während die angegebenen Werte somit die Einstellungen des ersten (i)FFT Beschleunigers darstellen, wird der zweite (i)FFT Beschleuniger automatisch zum Lesen an Schnittstelle 6 und zum Schreiben an Schnittstelle 7 angeschlossen. Seine Konfigurationsspeicheradresse wird auf 0xB0100000 gesetzt.

Die letzte Kategorie der System Generator GUI bietet die Möglichkeit, verschiedene Maximalwerte zu setzen, um Einfluss auf den Kompromiss zwischen Ressourcenverbrauch und Flexibilität zur Laufzeit zu nehmen. Auch können hier Voreinstellungen für einen Sender, einen Empfänger oder aber für einen Transceiver geladen werden. Die Checkbox *Fast Startup* implementiert den beschleunigten Systemstart nach Kapitel 2.2.7.2 für Empfängerdesigns, siehe Kapitel 7.2.3 für Details.

Sind die gewünschten Einstellungen vorgenommen, wird durch Betätigen des *Proceed* Knopfes die Konfigurationsdatei mit den VHDL-Generics sowie eine *system_parameters.h* Datei mit den Informationen als *#define* Direktiven, erstellt. Ers-

teres wird zur korrekten Erstellung des FPGA-Designs, letzteres zur korrekten Funktion der Software-Bibliothek benötigt.

Abschließend befindet man sich wieder im Entwicklungswerkzeug des Herstellers, wo durch ein Betätigen des Knopfes *Generate Programming File* der Bitstrom erstellt werden und durch ein Betätigen des Knopfes *Configure Target Device* der FPGA konfiguriert werden kann, siehe Abbildung 7.4.

Um den Bitstrom für einen beschleunigten Systemstart zu erstellen muss nach dem regulären Durchlauf noch zusätzlich *fast_startup.tcl* in die ISE-Konsole eingetragen werden. Es beginnt ein neuer Durchlauf des Designs, am Ende finden sich die zwei partiellen Bitströme *PI.bit* und *PII.bit* im Projektverzeichnis.

7.2.3. Referenzdesigns des System Generators

Zu Testzwecken wurden neben den aus Kapitel 5 bekannten Prototypen auch zwei neue Designs erzeugt, ein Transceiverdesign mit lediglich zwei SPS-Streams sowie eine energie- und ressourcenschonende Variante eines Empfängers. Letztere wurde dem Prinzip der Espresso Plattform [35] nachempfunden und wird daher im Folgenden *Espresso-Klon* genannt. Der Ressourcenverbrauch beider Designs ist in Tabelle 7.2 wiedergegeben, die Werte stammen aus den Angaben, welche nach der Platzierung und der Verdrahtung verfügbar sind.

Tabelle 7.2.: Ressourcenverbrauch weiterer, durch den System Generator erzeugte Designs.

Design Typ	Ressourcentyp								Max. Taktfreq.
	LUTs		Flip Flops		BRAMs		DSP		
Transceiver[a]	157 673	66%	115 684	24%	391	51%	247	28%	100 MHz
Espresso-Klon[b]	45 074	65%	33 884	36%	163	94%	132	100%	50 MHz

[a] Prozentangaben beziehen sich auf verfügbare Ressourcen eines XC6VHX380T
[b] Prozentangaben beziehen sich auf verfügbare Ressourcen eines XC6SLX100

Im 2-Stream Transceiverdesign wurden alle entwickelten Beschleuniger verwendet. Dabei wurde der (i)FFT-Beschleuniger zweimal pro Stream verwirklicht. Da man davon ausgehen kann, dass auch dieses Design wieder mit 125 MHz übertaktet werden kann, ist es in der Lage, gleichzeitig 500 MSa/s zu senden als auch zu empfangen. Da bei Fast Fourier Transformationen mit mehr als 32 Subträgern allerdings beide Beschleuniger eines Streams an der Transformation arbeiten müssen, ist bei diesen Größen ein gleichzeitiges Senden und Empfangen nicht

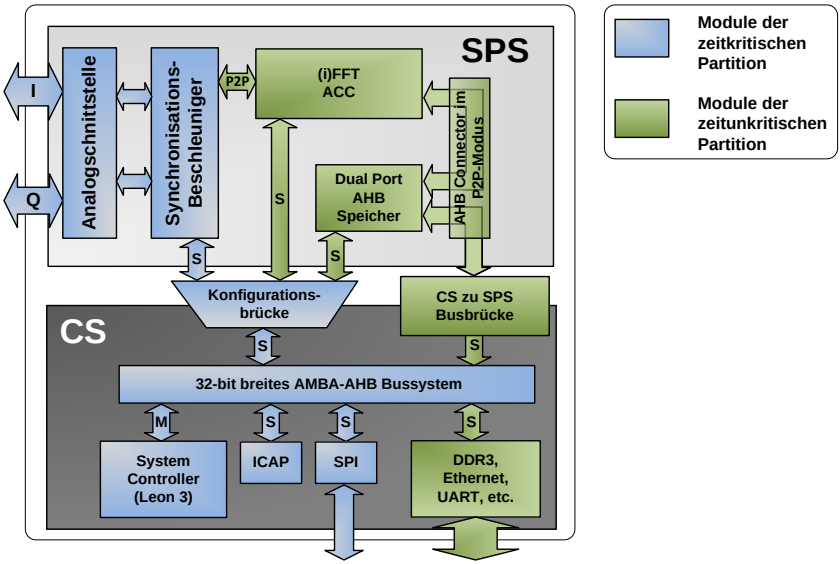


Abbildung 7.5.: Durch den System Generator erstellter Espresso-Klon: Der Hauptprozessor wird durch eine Analogschnittstelle, einen Synchronisationsbeschleuniger sowie einen Beschleuniger zur Berechnung der FFT ergänzt. Zusätzlich sind durch die angegebene Partitionierung schnelle Systemstarts per Fast Startup möglich.

mehr möglich. Tabelle 7.2 lässt jedoch erkennen, dass noch genügend Platz im Chip für weitere (i)FFT-Beschleuniger verfügbar ist.

Für das zweite Design, den Espresso-Klon, wurden einige Neuerungen vorgenommen. Es wurde nicht nur für den Virtex-6 FPGA, sondern auch für einen Spartan-6 (XC6SLX100) realisiert. Dieser Familie gehören die kleinen, energie- und kostensparenden FPGAs der Firma Xilinx an. Da diese FPGAs keine GTX-Transceiver besitzen, wurde die GTX-Brücke zur Adapterplatine entfernt und die Analogschnittstelle direkt in das Hauptdesign aufgenommen. Die Synthesewerkzeuge wurden so eingestellt, dass sie die Anzahl der Ressourcen optimieren und dass sie so viele DSP-Primitive wie möglich verwenden. Dies kann zusätzlich LUTs sparen.

Auch wurden für dieses Design die Techniken für einen schnellen Systemstart angewandt, vergleiche Kapitel 2.2.7. Dabei verwendet die Implementierung für Spartan-6 den eigens hierfür entwickelten Entwurfsfluss, siehe [MNH+11a]. Ab-

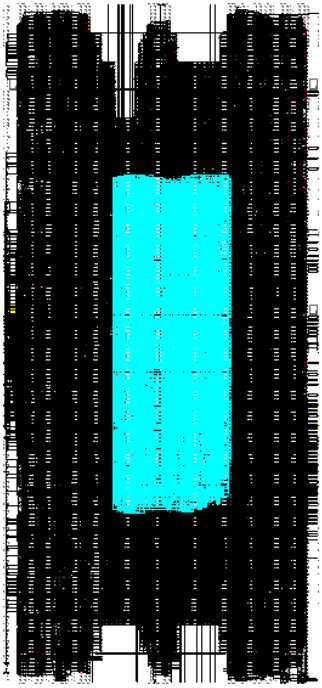


Abbildung 7.6.: FPGA Editor Ansicht des Espresso-Klon Designs in der Spartan-6 Variante. Netze der zeitkritischen Partition sind schwarz markiert, zeitunkritische Netze in Cyan dargestellt.

bildung 7.5 zeigt das Blockschaltbild des Systems und gliedert dabei die zeitkritischen und zeitunkritischen Komponenten farblich. Dies ermöglicht es dem System, schnell aus einem sehr tiefen Energiesparzustand aufzuwachen und Daten zu empfangen. Die CPU des CS kann zuerst die ADCs korrekt initialisieren und anschließend den Synchronisationsbeschleuniger programmieren. Während dieser dann bereits die ersten eingehenden Daten der ADCs nach einer Synchronisation durchsuchen kann, lädt die CPU des CS den partiellen Bitstrom der unkritischen Designpartition über die ICAP in den Konfigurationsspeicher.

Abbildung 7.6 zeigt das Spartan-6 Design in der Ansicht des FPGA Editors. Es ist gut zu erkennen, wie die Logik sowie die Netze der zeitunkritischen Partition (in Cyan markiert) streng auf einen möglichst großen Bereich festgelegt sind.

Somit ist der Bereich für die Ressourcen der zeitkritischen Partition (in schwarz) entsprechend auf eine kleine Fläche gedrängt.

Tabelle 7.3 listet die Bitstromgrößen der verschiedenen Bitstromvarianten und FPGAs. Da im Espresso-Klon Design ein Großteil der Komponenten zu der zeitkritischen Partition zählen, hält sich die Verringerung der Bitstromgröße im Vergleich mit einem komprimierten Bitstrom in Grenzen. Der Vergleich mit einem Standard-Bitstrom für den Spartan-6 FPGA fällt mit 69% dennoch bereits recht groß aus. Falls der prozentuale Anteil der zeitunkritischen Partition steigt, beispielsweise aufgrund eines Ausbaus des Espresso-Klon Design um weitere Beschleuniger, dann wird sich die prozentuale Beschleunigung durch Fast Startup ebenfalls entsprechend erhöhen.

Tabelle 7.3.: Vergleich der verschiedenen Bitstromgrößen in Bit für das Espresso-Klon Design

Bitstrom-Typ	**XC6SLX100**		**XC6VHX380T**	
Standard Bitstrom[a]	26 564 040	100%	119 785 488	100%
Komprimierter Bitstrom[a]	24 614 504	93%	45 170 768	38%
Komprimierter Bitstrom[b]	22 930 200	86%	28 073 936	23%
Fast Startup Bitstrom[b]	18 267 992	69%	20 456 496	17%

[a] Vollständiges Design
[b] Zeitkritische Komponenten im Sinne von Fast Startup

Gerade für große FPGAs und langsame Konfigurationsschnittstellen bestimmt die Bitstromgröße den Zeitbedarf der Konfigurationsphase , vgl. Formel 2.28 sowie die Werte des Espresso-Klons für Virtex-6. Dementsprechend wird diese Technik mit zukünftigen FPGAs eine immer größere Rolle spielen.

7.3. Softwareerstellung mit Hilfe des Frameworks

Da das Programmieren der Beschleuniger dieses Systems ohne Hilfsmittel ein detailliertes Wissen über dessen Aufbau und die Funktionsweise voraussetzen würde, wurde eine Softwarebibliothek erstellt, welche alle wichtigen Möglichkeiten zu Programmierung in einfache, dokumentierte Treiberfunktionen kapselt.

7.3.1. Organisation der Softwarebibliothek

Die umfangreiche Software-Bibliothek stellt auf verschiedenen Abstraktionsebenen Treiberfunktionen zur Verfügung. Abbildung 7.7 stellt diese drei Ebenen bild-

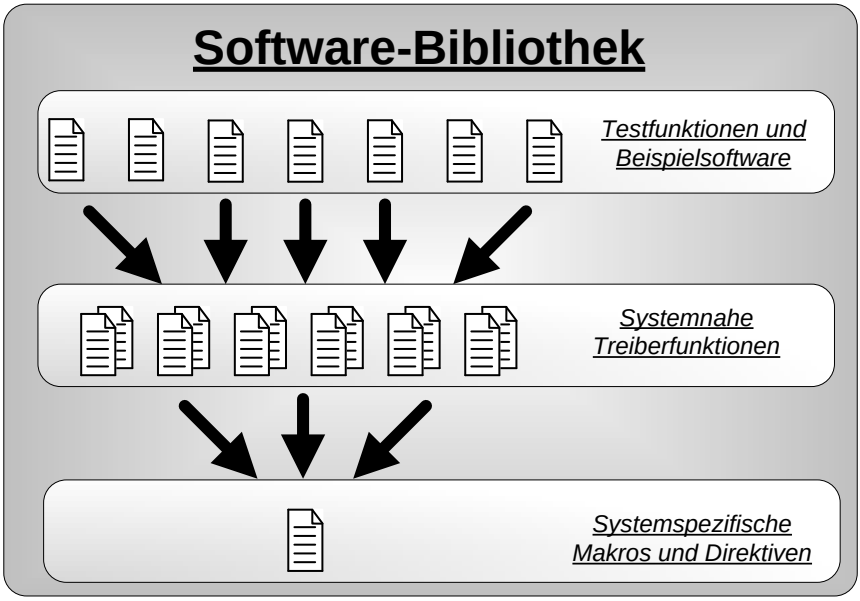


Abbildung 7.7.: Die Software Bibliothek ist in drei Hierarchieebenen geteilt. Während die unterste Ebene sehr systemnah ist, befinden sich in der oberen Ebene komplexe Testfunktionen und Beispielsoftware.

lich dar. Auf unterster Ebene werden die Konfigurations- und Befehlsspeicheradressen der Hardwarebeschleuniger in aussagekräftige Namen abgebildet und einfache Makros für Lese- und Schreibzugriffe auf die Register definiert. Zusätzlich werden hier benötigte Informationen des Designs, wie beispielsweise die Frequenz des Systemtaktes, zentral in der Datei *system_parameter.h* verwaltet und allen höheren Funktionen zur Verfügung gestellt. Diese Datei wird bei Verwendung des System Generators neu erstellt, damit die Informationen und die aktuelle Architektur konsistent sind.

Aufbauend auf der beschriebenen, untersten Treiberschicht existieren für jeden Beschleuniger Treiberdateien mit einer Vielzahl an Treiberfunktionen, welche die ordnungsgemäße Programmierung eines jeden Beschleunigers vereinfachen. So wird die Modifikation jeder Eigenschaft eines Beschleunigers durch mindestens eine Treiberfunktion angesprochen, oft auch in mehreren (beispielsweise zum Ein- und Ausschalten von Eigenschaften oder zum Schreiben und Lesen von Präambelwerten).

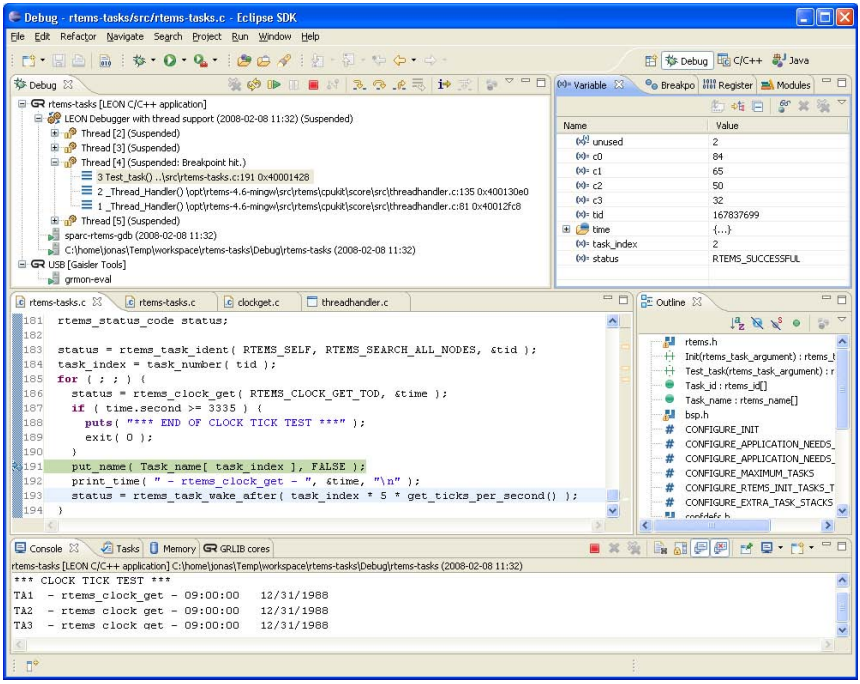


Abbildung 7.8.: Die Debugging-Ansicht in Eclipse. [44]

So existieren Funktionen, um einen Beschleuniger zu aktivieren oder deaktivieren, um ein Kommando in den Befehlsspeicher des Beschleunigers zu schreiben oder auch um die Kommandos von dort zu lesen. Ebenso lässt sich der Interruptmodus des Beschleunigers setzen oder lesen, sowie viele weitere Eigenschaften aktivieren oder deaktivieren. Um mit diesen Treiberfunktionen zu arbeiten, wird aufgrund der Vielzahl an Funktionen eine Einarbeitungszeit benötigt. Da solch ein Konzept im Embedded System Bereich allerdings verbreitet ist (vgl. Xilinx XPS) und die Softwarebibliothek zudem noch mit Hilfe des Software Dokumentationswerkzeuges *Doxygen* [38] ausführlich dokumentiert ist, fällt diese „Lernzeit“ eher gering aus.

Als letzte Schicht existieren zusätzlich komplexere Funktionen, wie beispielsweise Debugfunktionen. So kann bei Sendern mit einer entsprechenden Funktion die Ausgabe einer Sägezahnschwingung aktiviert werden, um die Hardware zu testen. Als weiteres Beispiel kann durch eine weitere Funktion die vollständige Ausgabe der aktuellen Konfiguration jedes Beschleunigers vorgenommen wer-

den. Auch eine Präambel zur Synchronisation und zur Kanalkorrektur lässt sich automatisch erzeugen und in den entsprechenden Präambelspeicher schreiben. All dies erleichtert den Aufbau eines Systems sowie eine eventuell notwendige Fehlersuche.

Ebenso existieren Testfunktionen für jeden Beschleuniger, durch welche gegebenenfalls deren Funktionalität überprüft werden kann. So ist es beispielsweise möglich, mit Hilfe der CS zu SPS Brücke vordefinierte Rohdaten in die Speicher des SPS zu schreiben, zu modulieren und anschließend mittels (i)FFT-Beschleunigern zu transformieren. Dabei werden die Ergebnisse durchgehend über die Brücke durch den Prozessor im CS auf Korrektheit überprüft. Bei unerwarteten Ergebnissen kann genau bestimmt werden, welcher Beschleuniger falsche Berechnungen durchführt.

Diese Funktionen und Beispielprogramme sind in Eclipse-Projekten organisiert. Eclipse ist eine fortschrittliche Programmierumgebung und bietet mit Hilfe des Plugins von Aeroflex-Gaisler auch Unterstützung für den Leon-3 Prozessor. So lässt sich die Software für ein System, welches mit Hilfe des Frameworks aufgebaut wurde, nicht nur innerhalb einer modernen integrierten Entwicklungsumgebung schreiben. Es unterstützt auch das grafische Debugging mit Hilfe des GNU Debugger, welcher alle üblichen Methoden zur Ablaufverfolgung unterstützt. Abbildung 7.8 zeigt ein Leon-3 Programm in der Debugging-Ansicht in Eclipse. Für weitere Details sei auf [44] verwiesen.

8. Schlussfolgerung und Ausblick

8.1. Schlussfolgerung

Mit dem HW/SW Co-Design Framework für Hochgeschwindigkeits-OFDM Signalverarbeitung wurde ein Baukastensystem geschaffen, mit welchem sich sehr leicht eine breite Palette an OFDM Systemen abdecken lässt. Eine Leistungsfähigkeit im Gb/s-Bereich, welche mit Hilfe des Systems und mit aktuellen FPGAs der Virtex-6 Familie erreicht wurde, war bisher nur mit Hilfe von dedizierten Systemen möglich.

Während dedizierte Systeme allerdings hohe Kosten durch eine lange Entwicklungszeit und durch die Notwendigkeit von Experten des Hardware-Entwurfs besitzen, kann ein entsprechendes, auf das Framework basiertes System, auf einfachste Weise realisiert werden. Das Framework und die grafische Benutzerschnittstelle ermöglicht es hierbei selbst Laien die Konfiguration eines Systems anzupassen und somit bereits zur Designzeit großen Einfluss auf die Möglichkeiten eines Hardware/Software Co-Design beziehungsweise auf den Kompromiss zwischen maximaler Datenrate und dem Ressourcenverbrauch zu nehmen.

Im Gegensatz zu dedizierten Systemen lässt sich ein mit Hilfe des Framework erstelltes System in allen wichtigen Parametern durch Programmierung schnell und einfach anpassen. Hierdurch kann es für eine enorme Vielzahl an Anwendungen verwendet werden. Es ist nicht nur möglich, durch einfaches Austauschen der Software einen anderen Industriestandard abzudecken, es können auch in Laboren und bei Feldtests auf einfache Weise und automatisiert die verschiedensten Parameter-Kombinationen durchlaufen werden.

Erweiterungen des Frameworks sind durch den modularen Aufbau und durch die Verwendung von Industriestandards leicht möglich. So können jegliche Arten von Prozessoren, Beschleunigern oder sonstigen Komponenten hinzugefügt werden, welche Alternativen zu den in dieser Arbeit verwendeten Algorithmen bieten oder weitere Funktionalitäten aus der Kommunikationstechnik, beispielsweise eine Fehlerkorrektur, zur Verfügung stellen.

Durch den im System vorhandenen General Purpose Prozessor des CS ist ein erstelltes System in der Lage, unter Reduzierung der erreichbaren Datenrate, per Software an jeder Stelle der Signalverarbeitungskette einzugreifen. Dies ist nicht

nur zu Debuggingzwecken sinnvoll, es können auch auf sehr schnelle und einfache Art und Weise neue Verarbeitungsschritte integriert werden. Anhand dieser Integration per Software kann anschließend evaluiert werden, ob sich der Entwurf eines entsprechenden Hardwarebeschleunigers lohnt.

Die rein verhaltensbasierte Beschreibung der Hauptkomponenten des Framework ermöglicht prinzipiell den Einsatz verschiedener Zieltechnologien. So sind die erzeugten Systeme des System Generators auch mit FPGAs anderer Hersteller realisierbar, allerdings hierfür weder getestet noch optimiert. Genauso wie für eine mögliche Realisierung mittels Standardzellen müssen hierzu noch Konzepte zur Technologieunabhängigkeit für Takt-, Speicher- und IO-Komponenten integriert werden. Ein Beispiel, wie dies bewerkstelligt werden kann, ist durch die GRLIB gegeben, vgl. [3].

Der Aufbau der aktuell durchsatzstärksten *software defined OFDM-Systeme* sowie die erfolgreichen Messungen zur Sende- und Empfangsqualität dieser Prototypen demonstrieren erfolgreich die Leistungsfähigkeit des Frameworks. Neben Datenraten im einstelligen Gigabit-Bereich konnten diese Experimente zudem die Flexibilität des Systems demonstrieren, indem eine Vielzahl an OFDM-Konfigurationen eingesetzt wurde.

In Kombination mit dem zusätzlich entwickelten und realisierten Digital Down Converter, welcher eine bisher unerreichte Bandbreite bietet, konnte leicht ein Szenario aktueller Forschungen im Umfeld des Radio-Backhauling realisiert werden. Dies verdeutlicht, wie flexible Kommunikationssysteme, die durch Software konfigurierbar und deren Parameter anpassbar sind, aber dennoch höchste Datenraten bieten, gewinnbringend im Forschungs- und Entwicklungsumfeld eingesetzt werden können.

Auch im industriellen Sektor bedeutet Flexibilität ein hoher Zugewinn, da sich Architekturen hierdurch unter Umständen für mehrere Anwendungen nutzen lassen, was die Stückzahlen erhöhen und die Kosten senken kann. Gerade durch das Konzept des Frameworks, welches mittels einfacher Schnittstellen entsprechende Systeme einem großen Kreis von Anwendern zugänglich macht, eröffnen sich in vielen Bereichen neue Felder für einen entsprechenden Kompromiss zwischen Flexibilität und maximaler Datenrate.

8.2. Ausblick

Da innerhalb dieser Arbeit das Hauptaugenmerk auf dem genannten Kompromiss von Durchsatz und Flexibilität liegt, wurde eine Optimierung des Energieverbrauchs nicht verfolgt. Dennoch dürfte sich auch in diesem Bereich, im Vergleich zu ähnlich flexiblen Architekturen, Potential ergeben, weshalb eine Opti-

mierung und Analyse in diese Richtung eine mögliche Weiterführung der Arbeit darstellt. Als Ausgangsbasis kann hierzu bereits das *Espresso-Klon* Design dienen, vgl. Kapitel 7.2.3. Eine Realisierung als Standardzellenentwurf ist hier jedoch obligatorisch, um einen fairen Vergleich mit anderen Veröffentlichungen zu ermöglichen.

Eine weitere interessante Anwendung auf FPGA-Basis ist die Verwendung der dynamischen partiellen Rekonfiguration. Dies könnte als Erweiterung der bereits durchgeführten Integration von Fast Startup geschehen, die bereits im *Espresso-Klon* Design enthalten ist. So könnte beispielsweise der (i)FFT-Beschleuniger als getrennte Partition realisiert werden, um ihn bei Bedarf durch andere Beschleuniger auszutauschen. So könnte es weniger Zeit beanspruchen, einen Equalization-Beschleuniger in das System zu konfigurieren, mit ihm die Kanalschätzung und Korrektur vorzunehmen und anschließend wieder den (i)FFT-Beschleuniger zu laden, als die Kanalkorrektur durch den Prozessor des CS per Software durchzuführen.

Ein anderer Anwendungsfall für dynamisch partielle Rekonfiguration ist es, die Konfiguration und die Programmierung der Beschleuniger zu übernehmen. Dadurch könnten die entsprechenden Konfigurationsbusse und die zugehörige Logik aus dem System entfernt werden. Alternativ wäre ein Übergang für die Architektur der Beschleuniger auf unflexible, dedizierte Schaltungen. Bei Parameteränderungen des Systems zur Laufzeit könnten diese Schaltungen mittels dynamischer partieller Rekonfiguration entsprechend ersetzt werden. Nachteil eines solchen Konzeptes wäre beispielsweise die große Anzahl benötigter Beschleunigervarianten. Außerdem würde man damit die Option eines Standardzellenentwurfes aufgeben.

Abgesehen von den erwähnten neuen Ausrichtungen des Systems kann das Framework natürlich beliebig ergänzt werden. Zum einen können weitere Beschleuniger entwickelt werden, welche den Entwurfsraum für mit Hilfe des Frameworks erstellten OFDM-Systeme vergrößern. So wären im Bereich des (i)FFT-Algorithmus vielerlei zusätzliche Varianten denkbar, beispielsweise die Implementierung anderer Radix-Arten.

Auch die Unterstützung anderer Prototyping Plattformen und Analogwandler sollte in Erwägung gezogen werden, um den Entwurfsraum zu erweitern. Gerade für den Entwurf kleiner, energieeffizienter Systeme sollten auch andere Varianten des Control Systems betrachtet werden. Alternativen wären beispielsweise Systeme, welche auf dem Microblaze Prozessor von Xilinx oder auf dem NIOS II Prozessor von Altera basieren. Auch Mischsysteme aus Prozessorsubsystem und FPGA, wie beispielsweise die ZYNQ Systeme von Xilinx, bieten vielversprechende Realisierungsmöglichkeiten für das Framework, siehe [113].

A. Mircostrip Antialiasing-Filter

Standardmäßig wird bei digitalen Empfängern das Empfangssignal vor der Digitalisierung mittels eines oder mehrerer ADCs durch einen Tiefpassfilter in seiner Bandbreite begrenzt. Dadurch lässt sich sicherstellen, dass das Nyquist-Shannon-Abtasttheorem eingehalten wird. Gerade bei Mehrträgersystemen können diese Filter nicht nur dazu verwendet werden, hochfrequente Störungen und unerwünschte Mischprodukte zu entfernen. Bereits hier können bei Bedarf Trägerfrequenzen des Signals ausgeblendet werden, die im Rahmen eines Frequenzmultiplexings für ihren Empfänger nicht von Interesse sind.

A.1. Anforderungen

Für die Empfangsexperimente (siehe Kapitel 5.3) sollten Antialiasing-Filter eingesetzt werden, welche das Empfangssignal bei der halben Abtastfrequenz begrenzen. Da der Empfänger-Prototyp eine Abtastrate von 1 GSa/s besitzt, müssen die Filter demnach alle Frequenzanteile entfernen, welche höher als 500 MHz liegen.

Da die äußeren Subträger eines OFDM Signals meist nicht moduliert werden und auch die Experimente dieser Arbeit keine Ausnahme darstellen, wurde als Passbandbereich 0 bis 460 MHz festgesetzt, der Stoppband-Bereich sollte bei 500 MHz beginnen. Die Dämpfung sollte nach Möglichkeit im Passbandbereich 1 dB nicht überschreiten, im Sperrbereich sollte eine Dämpfung von mindestens 20 dB bestehen. Aus organisatorischen Gründen konnten keine kommerziellen Filter verwendet werden, weshalb das Design und die Herstellung der Filter selbst übernommen wurde.

A.2. Entwicklung und Herstellung

Die Entwicklung und Herstellung der Filter erfolgte mit dem *Filter DesignGuide* der Software *Advanced Design System (ADS) 2008 Update 2* von Agilent. ADS stellt eine Umgebung für die Simulation und Entwicklung von Schaltungen der Hochfrequenztechnik dar. So lässt sich beispielsweise die Dimensionierung von

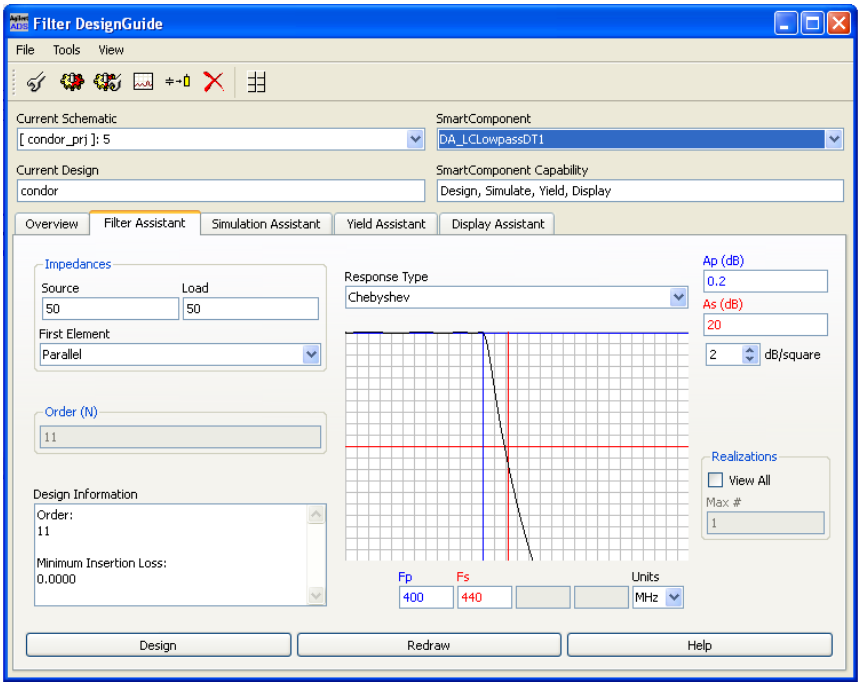


Abbildung A.1.: Mit Hilfe des *Filter DesignGuide* der Software *Advanced Design System (ADS) 2008 Update 2* von Agilent kann leicht eine Entwurfsraumexploration zur Filtererstellung stattfinden.

Leiterbahnen berechnen, um mittels Mikrostreifenleitungen Komponenten oder Schaltungen (wie beispielsweise Filter) zu erstellen.

Zuerst wurden die gewünschten Filtereigenschaften im *Filter DesignGuide* spezifiziert, wobei die maximal erlaubte Welligkeit auf 0,2 dB gesetzt wurde, da eine Degradierung durch Realbedingungen und Herstellungstoleranzen zu erwarten ist. Durch die Verwendung eines Tschebyscheff-Filters konnten wir schließlich unsere Anforderungen bereits mit einem Filter elfter Ordnung erreichen.

Abbildung A.1 zeigt das Fenster des Filter DesignGuide mit den eingestellten Werten des finalen Filterentwurfs. Dabei wurde ein Passband bis 400 MHz gewählt, das Stoppband beginnt entsprechend bei 440 MHz. Die Werte wurden korrigiert, da sich diese im späteren Verlauf des Designs noch entsprechend änderten.

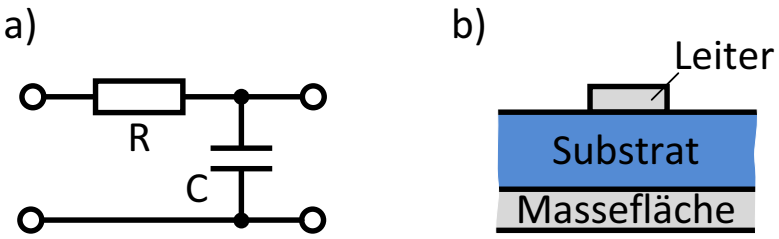


Abbildung A.2.: a) Einfacher Tiefpass mittels RC-Glied. b) Querschnitt durch eine Mikrostreifenleitung.

Die hier verwendeten Filter wurden mittels Mikrostreifenleitungen aufgebaut. Dabei handelt es sich um einfache Filter auf Basis von RC-Gliedern (vgl. Abbildung A.2 a)), wobei Widerstände durch Leiterbahnen realisiert werden und Kondensatoren durch Flächen, welche zwischen Leiter und Massenfläche eine Art Plattenkondensator bilden, vgl. Abbildung A.2 b). Für weiterführende Informationen sei auf [23] verwiesen.

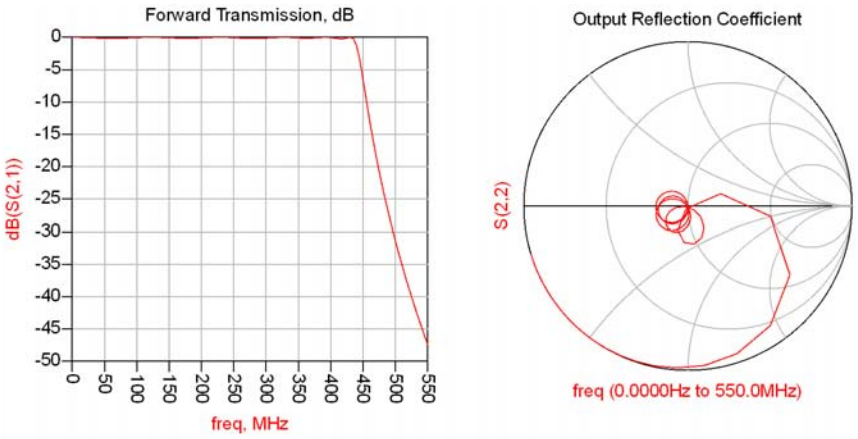


Abbildung A.3.: Ergebnisse einer Filtersimulation für die Vorwärts-Transmission und die Eingangsreflexion.

Im nächsten Schritt fand der genaue Filterentwurf statt. Von diesem Designschritt an werden alle Parameter des Platinenmaterials miteinbezogen, ebenso kann der

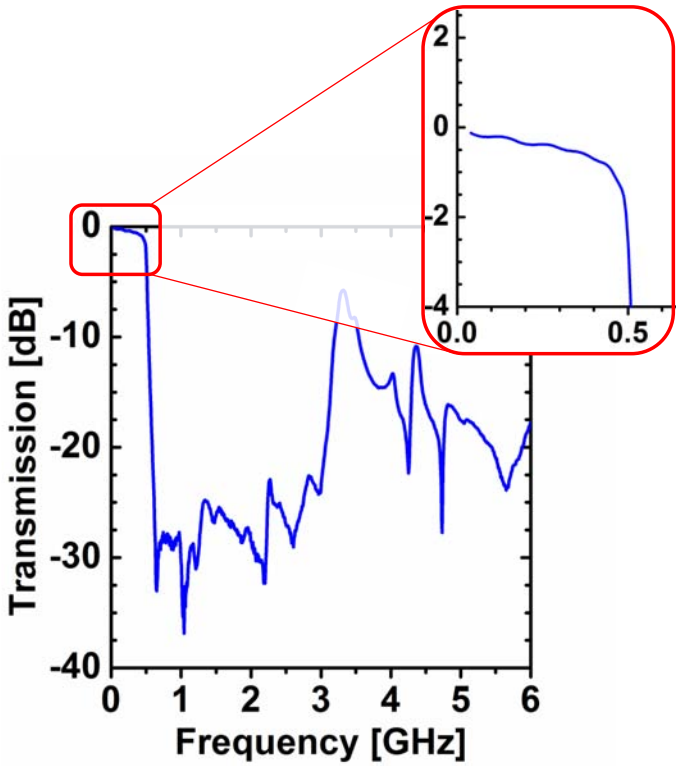


Abbildung A.4.: Frequenzverlauf eines Filters gemessen mit einem VNA von Anritsu (37397C).

Designer die Dimensionierung und den Verlauf der Leiterbahnen festlegen. Der Ein- und Ausgang des Filters wurde hierbei an SMA-Buchsen angebunden, wobei zu jedem Zeitpunkt im Entwurf Simulationen erstellt werden konnten.

Anhand der Simulationen wurden die Dimensionierung und die Eigenschaften der Leiterbahnen so lange verändert, bis das Ergebnis den Anforderungen entsprach und das Layout gleichzeitig platzsparend war, damit die Filter nicht zu viel Platinenmaterial benötigten und sich deren Einsatz in einem Laboraufbau angenehmer gestaltet.

Abbildung A.3 zeigt das Ergebnis einer Simulation des Filters in der endgültigen Fassung. Das Passband endet hier schon bei 430 MHz. Dies ist gewollt, da sich

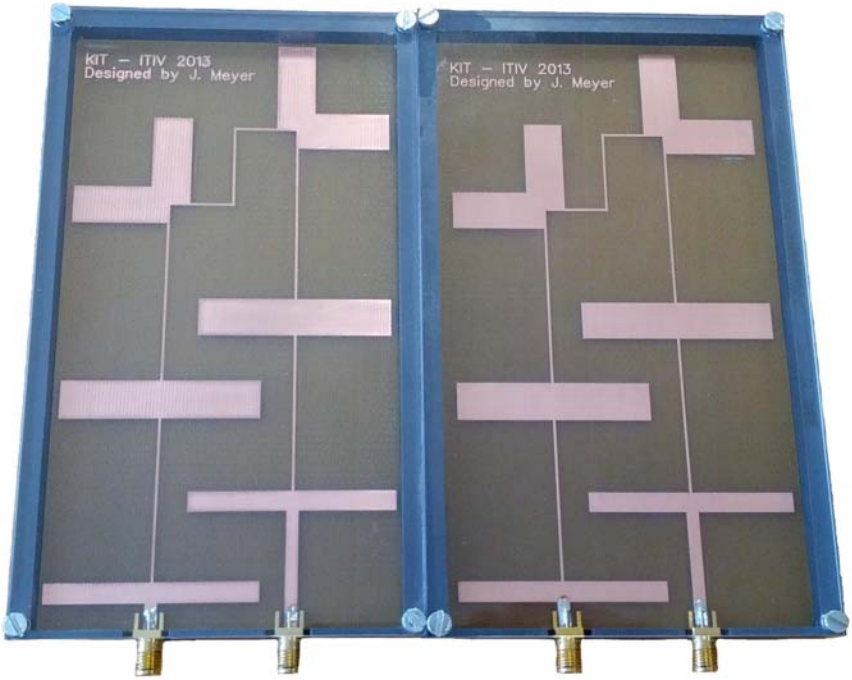


Abbildung A.5.: Zwei aufgebaute Filter in eigens angefertigtem PVC-Gehäuse.

bei einem Probelauf der Platinenproduktion nach dem Ausmessen mittels eines Netzwerkanalysators (Vector Network Analyzers, VNA) der Firma Anritsu herausstellte, dass mit den zur Verfügung stehenden Materialien und Techniken zur Platinenherstellung die von uns produzierten Filter später schließen als simuliert.

Die Messungen des finalen Filters mittels VNA sind in Abbildung A.4 dargestellt. Man kann erkennen, dass der Filter die Anforderungen in den meisten Punkten erfüllt, die 20 dB Dämpfung allerdings erst leicht über 500 MHz erreicht werden. Für unsere Experimente im Labor war dies ausreichend, da wir ausschließen konnten, dass sich entsprechende problematische Frequenzen im Signal befinden. Zwei produzierte und in PVC Gehäuse montierte Filter sind in Abbildung A.5 dargestellt.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

3GPP	3rd Generation Partnership Project
ACC	(HW-) Accelerator
ADC	Analog-to-Digital-Converter
ADS	Advanced Design System
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction-Set Processor
ASK	Amplitude-Shift Keying
AWG	Arbitrary Waveform Generator
AXI	Advanced eXtensible Interface
BDTI	Berkeley Design Technology, Inc.
BER	Bit error rate
BPI	Byte Peripheral Interface
BPSK	Binary phase-shift keying
BRAM	Block Random Accessible Memory
BSD	Berkeley Software Distribution
CIC	Cascaded-Integrator-Comb
CLB	Configurable Logic Block
COMP	Coordinated Multi-Point
CP	Cyclic Prefix
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CS	Control System
CUDA	Compute Unified Device Architecture
DAC	Digital-to-Analog Converter
DC	Direct Current
DCF77	Zeitzeichensender im Ortsteil Mainflingen in Mainhausen
DDC	Digital Down Converter
DDR	Double Data Rate
DFT	Diskreten Fourier-Transformation
DIF	Decimation in Frequency

DIT	Decimation in Time
DMA	Direct Memory Access
DP-BRAM	Dual Port BRAM
DSP	Digital Signal Processor
DSU3	Debug Support Unit 3
DVB-T	Digital Video Broadcasting - Terrestrial
E-B2B	Electrical Back-to-Back
EQ	Equalisation
EVM	Error-Vector-Magnitude
FDM	Frequency-Division Multiplexing
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FIFO	First-In First-Out Memory
FIR	Finite Impulse Response
FMC	FPGA Mezzanine Card
FPGA	Field Programmable Gate Array
FSK	Frequency-shift keying
GPGPU	General Purpose Computation on Graphics Processing Unit
GPP	General Purpose Processors
GPU	Graphics Processing Unit
GRLIB	Quelloffene VHDL IP Bibliothek von Gaisler Research
GUI	Graphical User Interface
GUIDE	Graphical User Interface Development Environment
HDL	Hardware Description Language
HW	Hardware
I	In-phase component
I/O, IO	Input/Output
I&Q, I/Q, IQ	in-phase and quadrature component
IC	Integrated Circuit
ICAP	Internal Configuration Access Port
ICI	Inter-carrier interference
ID	Identifier
IDFT	Inverse Diskreten Fourier-Transformation
IEEE	Institute of Electrical and Electronics Engineers
IF	Interface
IFFT	Inverse Fast Fourier Transform
IP	Internet Protocol
IP-Core	Intellectual property core
IRQ	Interrupt Request
ISE	Integrated Synthesis Environment
ISI	Intersymbol interference

JTAG ..... Joint Test Action Group
LD ..... Laserdiode
LTE ..... Long Term Evolution
LUT ..... Look-Up Table
LVDS ..... Low Voltage Differential Signaling
MAC ..... Medium Access Control
MC ..... Multi carrier
MIMO ..... Multiple Input Multiple Output
MPSoC ..... Multiprocessor System-on-Chip
NCO ..... Numerically Controlled Oscillator
NFC ..... Near Field Communication
NOC ..... Network-on-Chip
O-B2B ..... Optical Back-to-back
OFDM ..... Orthogonal Frequency-Division Multiplexing
OFDMA ..... Orthogonal Frequency-Division Multiple Access
P2P ..... Punkt-zu-Punkt Modus
PC ..... Personal Computer
PCIe ..... Peripheral Component Interconnect Express
PLB ..... Phase-locked loop
PLB ..... Processor Local Bus
PLC ..... Powerline Communication
PRBS ..... Pseudorandom Binary Sequence
PSK ..... Phase-shift keying
Q ..... Quadrature component
QAM ..... Quadrature amplitude modulation
QPSK ..... Quadrature phase-shift keying
RC-Glied ..... Resistor-Capacitor-Glied
RISC ..... Reduced Instruction Set Computer
RTL ..... Register Transfer Level
SC ..... Single carrier
SC-FDM ..... Single-carrier Frequency-Division Multiplexing
SDRAM ..... Static random-access memory
SDRAM ..... Synchronous Dynamic Random Access Memory
SIMD ..... Singe-Instruction-Multiple-Data
SMA ..... Sub-Miniature-A
SMF ..... Single-Mode Fibre
SOC ..... System-on-Chip
SPI ..... Serial Peripheral Interface
SPS ..... Signal Processing System
SW ..... Software
T-DMB ..... Terrestrial - Digital Multimedia Broadcasting

T-DMB	Terrestrial Digital Multimedia
TOD	Tunable Optical Delay
UART	Universal Asynchronous Receiver Transmitter
UMTS	Universal Mobile Telecommunications System
UWB	Ultra-wideband
VHDL	Very High Speed Description Language
VNA	Vector Network Analyzer
XPS	Xilinx® Platform Studio
XST	Xilinx® Synthesis Technology

Literatur- und Quellennachweise

[1] *LME2980 OFDM Powerline Communication IC datasheet*.

[2] ABDELALL, M., A. SHALASH und A. FAHMY: *A reconfigurable baseband processor for wireless OFDM synchronization sub-system*. In: *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, S. 2385–2388, May 2011.

[3] AEROFLEX GAISLER: *GRLIB IP Library User's Manual*, 1.3.7 Aufl., April 2014.

[4] AGMON, A., M. NAZARATHY, D. M. MAROM, S. BEN-EZRA, A. TOLMACHEV, R. KILLEY, P. BAYVEL, L. MEDER, M. HÜBNER, W. MEREDITH, G. VICKERS, P. C. SCHINDLER, R. M. SCHMOGROW, D. HILLERKUSS, W. FREUDE und J. LEUTHOLD: *Bi-directional Ultra-dense Polarization-muxed/diverse OFDM/WDM PON with Laserless Colorless 1Gb/s ONUs Based on Si PICs and <417 MHz Mixed-Signal ICs*. In: *Optical Fiber Communication Conference/National Fiber Optic Engineers Conference 2013*, S. OTh3A.6. Optical Society of America, 2013.

[5] ARM: *AMBA Specification (Rev 2.0)*, 1999.

[6] ARM: *Multi-layer AHB Overview*, B Aufl., 2004.

[7] ARM: *AMBA 3 AHB-Lite Protocol*, 1.0 Aufl., 2006.

[8] AZAR, C., M. OJAIL, S. CHEVOBBE und R. DAVID: *CERA: A Channel Estimation Reconfigurable Architecture*. In: *Telecommunications (ICT), 2010 IEEE 17th International Conference on*, S. 957–964, April 2010.

[9] BEEK, J.-J. VAN DE, O. EDFORS, M. SANDELL, S. WILSON und P. OLA BORJESSON: *On channel estimation in OFDM systems*. In: *Vehicular Technology Conference, 1995 IEEE 45th*, Bd. 2, S. 815–819 vol.2, Jul 1995.

[10] BEEK, J.-J. VAN DE, M. SANDELL und P. BORJESSON: *ML estimation of time and frequency offset in OFDM systems*. Signal Processing, IEEE Transactions on, 45(7):1800–1805, 1997.

[11] BENAVIDES, T., J. TREON, J. HULBERT und W. CHANG: *The Implementation of a Hybrid-Execute-In-Place Architecture to Reduce the Embedded System Memory Footprint and Minimize Boot Time*. In: *Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on*, S. 473–479, 2007.

[12] BERGLAND, G.: *Fast Fourier transform hardware implementations–An overview*.

Audio and Electroacoustics, IEEE Transactions on, 17(2):104–108, June 1969.

[13] BERKELEY DESIGN TECHNOLOGY, I.: *BDTI OFDM Receiver Benchmark™Certified Results*. http://www.bdti.com/Resources/BenchmarkResults/OFDM. Used with permission.

[14] BHATTACHARYYA, S. und E. DEPRETTERE: *Handbook of Signal Processing Systems*. Springer, 2010.

[15] BIERMANN, T.: *Dealing with backhaul network limitations in coordinated multipoint deployments*. Doktorarbeit, Universität Paderborn, 2012.

[16] BIRK, M., R. DAPP, N. RUITER und J. BECKER: *GPU-based iterative transmission reconstruction in 3D ultrasound computer tomography*. Journal of Parallel and Distributed Computing, 74:1730 – 1743, 2014.

[17] BROWN, S., R. FRANCIS, J. ROSE und Z. VRANESIC: *Field-Programmable Gate Arrays*. . VLSI, computer architecture and digital signal processing. Springer US, 1992.

[18] CHEN, Y.-M. und I.-Y. KUO: *Design of lowpass filter for digital down converter in OFDM receivers*. In: *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, Bd. 2, S. 1094–1099 vol.2, June 2005.

[19] CHU, E. und A. GEORGE: *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*. Computational Mathematics. Taylor & Francis, 1999.

[20] CISCO SYSTEMS, I.: *The Zettabyte Era - Trends and Analysis*. online: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.pdf, Juni 2014.

[21] COOLEY, J. W. und J. W. TUKEY: *An Algorithm for the Machine Calculation of Complex Fourier Series*. Mathematics of Computation, 19(90):pp. 297–301, 1965.

[22] CZYLWIK, A.: *Downlink beamforming for mobile radio systems with frequency division duplex*. In: *Personal, Indoor and Mobile Radio Communications, 2000. PIMRC 2000. The 11th IEEE International Symposium on*, Bd. 1, S. 72–76 vol.1, 2000.

[23] DAS, S.: *The Basic Idea of Microstrip Filter Design for Wireless Communication*. LAP Lambert Academic Publishing, 2012.

[24] DINECHIN, F. DE, H. TAKEUGMING und J.-M. TANGUY: *A 128-tap complex FIR filter Processing 20 giga-samples/s in a single FPGA*. In: *Signals, Systems and Computers (ASILOMAR), 2010 Conference Record of the Forty Fourth Asilomar*

Conference on, S. 841–844, Nov 2010.

[25] DRESCHMANN, M.: *ACCORDANCE Deliverable*. Projektbericht 3.6, Karlsruher Institut für Technologie (KIT), September 2013.

[26] DURAI, R.: *Digital Signal Processing*. Laxmi Publications, 2005.

[27] FAYEZ, A., N. KAMINSKI, A. YOUNG und C. BOSTIAN: *Embedded SDR system design case study: An implementation perspective*. In: *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, S. 1–6, June 2012.

[28] FISCHER, W.: *Digitale Fernsehtechnik in Theorie und Praxis: MPEG-Basiscodierung ; DVB-, DAB-, ATSC-Übertragungstechnik ; Messtechnik*. Springer, 2006.

[29] GIRARD, O.: *openmsp430 Projekt auf Open Cores: http://opencores.org*.

[30] GOODMAN, D. und M. CAREY: *Nine digital filters for decimation and interpolation*. Acoustics, Speech and Signal Processing, IEEE Transactions on, 25(2):121–126, Apr 1977.

[31] GRUIJTERS, P. und B. VANDEWIELE: *Algorithm Partitioning and SoC Design for OFDM Communication Systems Using Multiple Application Specific Processors*. In: *Multimedia, 2006. ISM'06. Eighth IEEE International Symposium on*, S. 481–487, Dec 2006.

[32] GUAN, X., Y. FEI und H. LIN: *Hierarchical Design of an Application-Specific Instruction Set Processor for High-Throughput and Scalable FFT Processing*. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 20(3):551–563, March 2012.

[33] HAENE, S., A. BURG, P. LUETHI, N. FELBER und W. FICHTNER: *FFT Processor for OFDM Channel Estimation*. In: *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, S. 1417–1420, May 2007.

[34] HAJJAR, C. E.: *Synchronization Algorithms for OFDM Systems (IEEE802.11a, DVB-T) : Analysis, Simulation, Optimization and Implementation Aspects*. Doktorarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2008.

[35] HARJU, L.: *Programmable Receiver Architecture for Multimode Mobile Terminals*. Doktorarbeit, Tampere University of Technology, 2006.

[36] HAUBELT, C. und J. TEICH: *Digitale Hardware/Software-Systeme*. eXamenpress. Springer, 2010.

[37] HE, H. und H. GUO: *The Realization of FFT Algorithm Based on FPGA Co-Processor*. In: *Intelligent Information Technology Application, 2008. IITA '08. Second International Symposium on*, Bd. 3, S. 239–243, Dec 2008.

[38] HEESCH, D. VAN: *doxygen Manual for version 1.8.7*, 2014.

[39] HSIEH, M.-H. und C.-H. WEI: *Channel estimation for OFDM systems based on comb-type pilot arrangement in frequency selective fading channels*. Consumer Electronics, IEEE Transactions on, 44(1):217–225, Feb 1998.

[40] INAN, B., S. ADHIKARI, O. KARAKAYA, P. KAINZMAIER, M. MOCKER, H. VON KIRCHBAUER, N. HANIK und S. JANSEN: *Realization of a real-time 93.8-Gb/s polarization-multiplexed OFDM transmitter with 1024-point IFFT*. In: *Optical Communication (ECOC), 2011 37th European Conference and Exhibition on*, S. 1–3, Sept 2011.

[41] INSTRUMENTS, T.: *GC4016 Multi-Standard Quad DDC Chip Data Manual*, Revision B Aufl., 2009.

[42] ITWISSEN.INFO: *Nahfeldkommunikation*. online: http://www.itwissen.info/definition/lexikon/Nahfeldkommunikation-NFC-near-field-communication.html, July 2014.

[43] JO, B. und M. SUNWOO: *New continuous-flow mixed-radix (CFMR) FFT Processor using novel in-place strategy*. Circuits and Systems I: Regular Papers, IEEE Transactions on, 52(5):911–919, May 2005.

[44] JONAS FLODÉN, K. E.: *LEON Integrated Development Environment for Eclipse*. Aeroflex Gaisler, 1.48 Aufl., December 2013.

[45] KAESLIN, H.: *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication*. Cambridge University Press, 2008.

[46] KAMMEYER, K., M. BOSSERT und N. FLIEGE: *Nachrichtenübertragung*. Vieweg+Teubner Verlag, 2011.

[47] KANEDA, N., Q. YANG, X. LIU, S. CHANDRASEKHAR, W. SHIEH und Y.-K. CHEN: *Real-Time 2.5 GS/s Coherent Optical Receiver for 53.3-Gb/s Sub-Banded OFDM*. Lightwave Technology, Journal of, 28(4):494–501, Feb 2010.

[48] KANONAKIS, K., I. TOMKOS, H. KRIMMEL, F. SCHAICH, C. LANGE, E. WEIS, J. LEUTHOLD, M. WINTER, S. ROMERO, P. KOURTESSIS, M. MILOSAVLJEVIC, I. CANO und O. PRAT: *An OFDMA-based optical access network architecture exhibiting ultra-high capacity and wireline-wireless convergence.* Communications Magazine, IEEE, 50(8):71–78, August 2012.

[49] KAUFFMAN, K., J. RAQUET, Y. MORTON und D. GARMATYUK: *Real-Time UWB-OFDM Radar-Based Navigation in Unknown Terrain*. Aerospace and Electronic Systems, IEEE Transactions on, 49(3):1453–1466, July 2013.

[50] KHATRI, S. und K. GULATI: *Hardware Acceleration of EDA Algorithms: Custom ICs, FPGAs and GPUs*. Springer, 2010.

[51] KUON, I. und J. ROSE: *Measuring the Gap Between FPGAs and ASICs*. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transac-

tions on, 26(2):203–215, 2007.

[52] KYLLIAINEN, J., J. NURMI und M. KUULUSA: *COFFEE - a core for free*. In: *System-on-Chip, 2003. Proceedings. International Symposium on*, S. 17–22, Nov 2003.

[53] LEE, H. G., N. CHANG, U. Y. OGRAS und R. MARCULESCU: *On-chip Communication Architecture Exploration: A Quantitative Evaluation of Point-to-point, Bus, and Network-on-chip Approaches*. ACM Trans. Des. Autom. Electron. Syst., 12(3):23:1–23:20, Mai 2008.

[54] LEE, J., Y. KIM, H. LEE, B. L. NG, D. MAZZARESE, J. LIU, W. XIAO und Y. ZHOU: *Coordinated multipoint transmission and reception in LTE-advanced systems*. Communications Magazine, IEEE, 50(11):44–50, November 2012.

[55] LEISERSON, C. E., F. M. ROSE und J. B. SAXE: *Optimizing Synchronous Circuitry by Retiming (Preliminary Version)*. In: BRYANT, R. (Hrsg.): *Third Caltech Conference on Very Large Scale Integration*, S. 87–116. Springer Berlin Heidelberg, 1983.

[56] LI, G. und G. STÜBER: *Orthogonal Frequency Division Multiplexing for Wireless Communications*. Signals and communication technology. Springer Science+Business Media, 2006.

[57] MA, X., H. ZHAO, G. LI und Y. ZHAO: *Implementation of a high-throughput OFDM system using Graphics Processing Units*. In: *Communication Technology (ICCT), 2013 15th IEEE International Conference on*, S. 639–644, Nov 2013.

[58] MALLADI, S., S. MYNENI, P. POTHANA und M. BAYOUMI: *A high speed pipelined FFT processor*. In: *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, S. 1609–1612 vol.3, Apr 1991.

[59] MICHEL, P., U. LAUTHER und P. DUZY: *The Synthesis Approach to Digital System Design*. . VLSI, computer architecture and digital signal processing. Springer US, 1992.

[60] MICRAM: *VEGA ADC30 Datasheet*, preliminary Aufl.

[61] MING, C., Q. HOUDE, Z. HUAN und L. WEI: *Design of High-Speed DDC Based Multi-stage*. In: *Pervasive Computing Signal Processing and Applications (PCSPA), 2010 First International Conference on*, S. 936–939, Sept 2010.

[62] MINN, H., V. BHARGAVA und K. LETAIEF: *A robust timing and frequency synchronization for OFDM systems*. Wireless Communications, IEEE Transactions on, 2(4):822–839, 2003.

[63] MINN, H., M. ZENG und V. BHARGAVA: *On timing offset estimation for OFDM systems*. Communications Letters, IEEE, 4(7):242–244, July 2000.

[64] MIZUOCHI, T.: *Recent progress in forward error correction and its interplay with*

transmission impairments. Selected Topics in Quantum Electronics, IEEE Journal of, 12(4):544–554, July 2006.

[65] MOOSE, P. H.: *A technique for orthogonal frequency division multiplexing frequency offset correction*. Communications, IEEE Transactions on, 42(10):2908–2914, 1994.

[66] MOSIER, R. R. und R. G. CLABAUGH: *Kineplex, a bandwidth-efficient binary transmission system*. American Institute of Electrical Engineers, Part I: Communication and Electronics, Transactions of the, 76(6):723–728, 1958.

[67] NATIONAL INSTRUMENTS: *Creating Custom Hardware with LabVIEW*, 2014. http://www.ni.com/pdf/products/us/labview_fpga_module.pdf.

[68] NEVES, N., N. SEBASTIAO, A. PATRICIO, D. MATOS, P. TOMAS, P. FLORES und N. ROMA: *BioBlaze: Multi-core SIMD ASIP for DNA sequence alignment*. In: *Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference on*, S. 241–244, June 2013.

[69] NYLANDEN, T., J. JANHUNEN, O. SILVEN und M. JUNTTI: *A GPU implementation for two MIMO-OFDM detectors*. In: *Embedded Computer Systems (SAMOS), 2010 International Conference on*, S. 293–300, July 2010.

[70] OBERSCHELP, W. und G. VOSSEN: *Rechneraufbau und Rechnerstrukturen*. Oldenbourg, 2006.

[71] OPENCORES: *Wishbone B4 - WISHBONE System-on-Chip (SoC)Interconnection Architecturefor Portable IP Cores*, 2010.

[72] PASRICHA, S. und N. DUTT: *On-Chip Communication Architectures: System on Chip Interconnect*. Systems on Silicon. Elsevier Science, 2010.

[73] PATTERSON, D. und J. HENNESSY: *Computer Organization and Design, Fourth Edition: The Hardware/Software Interface*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2008.

[74] PUENTE LEÓN, F. und U. KIENCKE: *Messtechnik - Systemtheorie für Ingenieure und Informatiker*. Springer-Verlag, Berlin Heidelberg, 8 Aufl., 2011.

[75] QIAN, D., T.-O. KWOK, N. CVIJETIC, J. HU und T. WANG: *41.25 Gb/s real-time OFDM receiver for variable rate WDM-OFDMA-PON transmission*. In: *Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC)*, S. 1–3, March 2010.

[76] REESE, J. und S. ZARANEK: *GPU Programming in MATLAB*. Techn. Ber., MathWorks, 2011.

[77] REINERT, M.: *FPGA-basierte Realisierung eines OFDM-Funkübertragungssystems*. Doktorarbeit, Technische Universität Hamburg-Harburg, 2009.

[78] RINNE, J. und M. RENFORS: *Pilot spacing in orthogonal frequency division multiplexing systems on practical channels*. Consumer Electronics, IEEE Transactions on, 42(4):959–962, Nov 1996.

[79] SCHABAS, K. und S. D. BROWN: *Using Logic Duplication to Improve Performance in FPGAs*. In: *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays*, FPGA '03, S. 136–142, New York, NY, USA, 2003. ACM.

[80] SCHMIDL, T. und D. COX: *Robust frequency and timing synchronization for OFDM*. Communications, IEEE Transactions on, 45(12):1613–1621, 1997.

[81] SCHMOGROW, R., M. MEYER, P. SCHINDLER, A. JOSTEN, S. BEN-EZRA, C. KOOS, W. FREUDE und J. LEUTHOLD: *252 Gbit/s real-time Nyquist pulse generation by reducing the oversampling factor to 1.33*. In: *Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC), 2013*, S. 1–3, March 2013.

[82] SCHÄUFFELE, J. und T. ZURAWKA: *Automotive Software Engineering*. Springer Verlag, 5 Aufl., 2013.

[83] SHAFIK, R., S. RAHMAN und R. ISLAM: *On the Extended Relationships Among EVM, BER and SNR as Performance Metrics*. In: *Electrical and Computer Engineering, 2006. ICECE '06. International Conference on*, S. 408–411, Dec 2006.

[84] SHEN, Y. und E. MARTINEZ: *Channel Estimation in OFDM Systems (AN3059)*. Application Note, Freescale Semiconductor, January 2006.

[85] SKALICKY, S., C. WOOD, M. LUKOWIAK und M. RYAN: *High level synthesis: Where are we? A case study on matrix multiplication*. In: *Reconfigurable Computing and FPGAs (ReConFig), 2013 International Conference on*, S. 1–7, Dec 2013.

[86] STADLER, E.: *Modulationsverfahren: Analoge und digitale Modulation in der elektrischen Nachrichtentechnik*. Kamprath-Reihe. Vogel Business Media, 8. Aufl., 2000.

[87] STURM, C., T. ZWICK und W. WIESBECK: *An OFDM System Concept for Joint Radar and Communications Operations*. In: *Vehicular Technology Conference, 2009. VTC Spring 2009. IEEE 69th*, S. 1–5, April 2009.

[88] SUI, D., Y. LI, J. WANG, P. WANG und B. ZHOU: *High throughput MIMO-OFDM detection with graphics processing units*. In: *Computer Science and Automation Engineering (CSAE), 2012 IEEE International Conference on*, Bd. 2, S. 176–179, May 2012.

[89] SUN, J., N. CEN und D. YUAN: *Implementation of a 2x2 MIMO-OFDM Real-Time System on DSP/FPGA Platform*. In: *Communications and Mobile Computing (CMC), 2011 Third International Conference on*, S. 441–444, April 2011.

[90] TEICHERT, F.: *Digital-Down-Converter für Field-Programmable-Gate-Arrays in VHDL*. Diplomarbeit, Department Informations- und Elektrotechnik der Fakultät Technik und Informatik der Hochschule für Angewandte Wissenschaften Hamburg, März 2009.

[91] TSENG, W.-H., C.-C. CHANG und C.-K. WANG: *Digital VLSI OFDM transceiver architecture for wireless SoC design*. In: *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, S. 5794–5797 Vol. 6, May 2005.

[92] WANG, W., L. LI, G. ZHANG, D. LIU und J. QIU: *An Application Specific Instruction Set Processor optimized for FFT*. In: *Circuits and Systems (MWSCAS), 2011 IEEE 54th International Midwest Symposium on*, S. 1–4, Aug 2011.

[93] WEISER, M.: *The computer for the 21st century*. SIGMOBILE Mob. Comput. Commun. Rev., 3(3):3–11, Juli 1999.

[94] WERNER, M. und O. MILDENBERGER: *Nachrichten-Übertragungstechnik: Analoge und digitale Verfahren mit modernen Anwendungen*. Studium Technik. Vieweg+Teubner Verlag, 2006.

[95] WINTERSTEIN, F., S. BAYLISS und G. CONSTANTINIDES: *High-level synthesis of dynamic data structures: A case study using Vivado HLS*. In: *Field-Programmable Technology (FPT), 2013 International Conference on*, S. 362–365, Dec 2013.

[96] XILINX: *Difference-Based Partial Reconfiguration (XAPP290)*, 2.0 Aufl., December 2007.

[97] XILINX: *Early Access Partial Reconfiguration User Guide (UG208)*, 1.2 Aufl., September 2008.

[98] XILINX: *Serial ATA Physical Link Initialization with the GTP Transceiver of Virtex-5 LXT FPGAs (XAPP870)*, 1.0 Aufl., Januar 2008.

[99] XILINX: *LogiCORE IP ChipScope Pro Integrated Logic Analyzer (ILA) (v1.04a)(DS299)*, March 2011.

[100] XILINX: *Spartan-6 FPGA Data Sheet: DC and Switching Characteristics (DS162)*, 3.0 Aufl., October 2011.

[101] XILINX: *Virtex-6 FPGA DSP48E1 Slice User Guide (UG369)*, 1.3 Aufl., February 2011.

[102] XILINX: *Virtex-6 FPGA GTH Transceivers User Guide (UG371)*, 2.2 Aufl., June 2011.

[103] XILINX: *Virtex-6 FPGA GTX Transceivers User Guide (UG366)*, 2.6 Aufl., July 2011.

[104] XILINX: *ChipScope Pro Software and Cores User Guide (UG029)*, 14.3 Aufl., October 2012.

[105] XILINX: *Virtex-6 Family Overview (DS150)*, 2.4 Aufl., Januar 2012.

[106] XILINX: *Virtex-6 FPGA Configurable Logic Block User Guide (UG364)*, 1.2 Aufl., February 2012.

[107] XILINX: *Command Line Tools User Guide (UG628)*, 14.7 Aufl., October 2013.

[108] XILINX: *Partial Reconfiguration User Guide (UG702)*, 14.5 Aufl., April 2013.

[109] XILINX: *Spartan-6 FPGA Configuration User Guide (UG380)*, 2.5 Aufl., Januar 2013.

[110] XILINX: *Virtex-6 FPGA Configuration User Guide (UG360)*, 3.7 Aufl., November 2013.

[111] XILINX: *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics (DS152)*, 3.5 Aufl., May 2013.

[112] XILINX: *Virtex-6 FPGA Memory Resources User Guide*, 1.7 Aufl., September 2013.

[113] XILINX: *Zynq-7000 All Programmable SoC Overview*, 1.6 Aufl., December 2013.

[114] XILINX: *LogiCORE IP Fast Fourier Transform (PG109)*, 9.0 Aufl., April 2014.

[115] XILINX: *Press Release: Xilinx Announces Record Fiscal 2014 Revenues (available at http://investor.xilinx.com/results.cfm)*. online, 2014.

[116] YAN, H., S. ZHOU, Z. SHI, J.-H. CUI, L. WAN, J. HUANG und H. ZHOU: *DSP implementation of SISO and MIMO OFDM acoustic modems*. In: *OCEANS 2010 IEEE - Sydney*, S. 1–6, May 2010.

[117] YEH, H.-G. und P. INGERSON: *Software-defined radio for OFDM transceivers*. In: *Systems Conference, 2010 4th Annual IEEE*, S. 261–266, April 2010.

[118] ZHANG, Q. und X. SU: *The Design of Digital Down Converter Based on FPGA*. In: *Wireless Communications, Networking and Mobile Computing (WiCOM), 2012 8th International Conference on*, S. 1–4, Sept 2012.

[119] ZHIQIANG, H., S. JIANXING, D. RAN und Y. CHEN: *Analysis for singal processing development with general purpose processor*. In: *Communications and Networking in China (CHINACOM), 2012 7th International ICST Conference on*, S. 792–796, Aug 2012.

Betreute studentische Arbeiten

[Fri10] FRIEDERICH, STEPHANIE: *Integration of a Custom Toolflows for FPGAs in Xilinx EDK*. Diplomarbeit ID-1487, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2010.

[Men12] MENZEL, SIMON: *Entwurf und Implementierung eines parallel arbeitenden Digital Down Converters zur Vorverarbeitung eines 25 GS/s OFDM-Signals*. Studienarbeit IL-981, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2012.

[Mer13] MERZ, CHRISTOPHER: *Aufbau einer parametrisierbaren Simulation eines OFDM Kommunikationssystems für einen optischen Kanal*. Bachelorarbeit ID-1660, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2013.

[Ruh11] RUHLAND, CHRISTOPH: *Design und Implementierung eines FPGA-basierten Highspeed OFDM Transmitters*. Diplomarbeit ID-1487, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2011.

[Sch12] SCHLANGEN, PATRICK: *Entwurf und Implementierung einer dynamisch rekonfigurierbaren elektronischen Steuereinheit auf Basis eines Kintex-7 FPGAs*. Bachelorarbeit ID-1621, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2012.

[Sto12] STORK, SEBASTIAN: *Performance Analysis for Emerging Memory Technologies*. Diplomarbeit ID-1607, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2012.

[Wit12] WITZIG, ALEXEJ: *Design und Implementierung eines System-On-Chip zur Signalverarbeitung in Highspeed OFDM-Transmittern*. Masterarbeit ID-1522, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2012.

[Wöl12] WÖLFLE, SANDRO-DIEGO: *Entwurf und Implementierung von Synchronisations- und Kanalkorrektur-Prozessoren für Hochgeschwindigkeits-OFDM*. Diplomarbeit ID-1560, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2012.

Eigene Veröffentlichungen

Konferenzbeiträge, Journalbeiträge und Buchkapitel

[DJS+14] DRESCHMANN, M., MEYER J., P. C. SCHINDLER, R. SCHMOGROW, J. LEUTHOLD, W. FREUDE und BECKER J.: *An ultra-high speed OFDMA system for optical access networks*. In: *Proceedings of International Conference on Computing, Networking and Communications (ICNC)*, February 2014.

[DMH+11] DRESCHMANN, M., J. MEYER, M. HUBNER, R. SCHMOGROW, D. HILLERKUSS, J. BECKER, J. LEUTHOLD und W. FREUDE: *Implementation of an ultra-high speed 256-point FFT for Xilinx Virtex-6 devices*. In: *9th IEEE International Conference on Industrial Informatics (INDIN)*, 2011.

[DMH+12] DRESCHMANN, M., J. MEYER, M. HUBNER, R. SCHMOGROW, D. HILLERKUSS, J. BECKER, J. LEUTHOLD und W. FREUDE: *Time and frequency synchronization for ultra-high speed OFDM systems*. In: *Proceedings of International Conference on Computing, Networking and Communications (ICNC)*, 2012.

[FHS+11] FREUDE, W., D. HILLERKUSS, T. SCHELLINGER, R. SCHMOGROW, M. WINTER, T. VALLAITIS, R. BONK, A. MARCULESCU, J. LI, M. DRESCHMANN, J. MEYER, S. BEN-EZRA, M. CASPI, B. NEBENDAHF, F. PARMIGIANI, P. PETROPOIU, B. RESAN, A. OEHLER, K. WEINGARTEN, T. ELLERMEYER, J. LUTZ, M. MOLLER, M. HUEBNER, J. BECKER, C. KOOS und J. LEUTHOLD: *All-optical real-time OFDM transmitter and receiver*. In: *Conference on Lasers and Electro-Optics (CLEO)*, 2011.

[FSD+12] FREUDE, W., R. SCHMOGROW, HILLERKUSS D., MEYER J., M. DRESCHMANN, B. NEBENDAHL, M. HUEBNER, BECKER J., C. KOOS und J. LEUTHOLD: *Reconfigurable optical transmitters and receivers*. Proceedings SPIE 8284, Next-Generation Optical Communication: Components, Sub-Systems, and Systems, 2012.

[FSN+11] FREUDE, W., R. SCHMOGROW, B. NEBENDAHL, D. HILLERKUSS, J. MEYER, M. DRESCHMANN, M. HUEBNER, J. BECKER, C. KOOS

und J. LEUTHOLD: *Software-defined optical transmission*. In: *13th International Conference on Transparent Optical Networks (ICTON)*, 2011.

[FSN+12] FREUDE, W., R. SCHMOGROW, B. NEBENDAHL, M. WINTER, A. JOSTEN, D. HILLERKUSS, S. KOENIG, J. MEYER, M. DRESCHMANN, M. HUEBNER, C. KOOS, J. BECKER und J. LEUTHOLD: *Quality metrics for optical signals: Eye diagram, Q-factor, OSNR, EVM and BER*. In: *14th International Conference on Transparent Optical Networks (ICTON)*, 2012.

[HMS+10] HUEBNER, M., J. MEYER, O. SANDER, L. BRAUN, J. BECKER, J. NOGUERA und R. STEWART: *Fast Sequential FPGA Startup Based on Partial and Dynamic Reconfiguration*. In: *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2010.

[HSS+10] HILLERKUSS, D., T. SCHELLINGER, R. SCHMOGROW, M. WINTER, T. VALLAITIS, R. BONK, A. MARCULESCU, J. LI, M. DRESCHMANN, J. MEYER, S. BEN EZRA, N. NARKISS, B. NEBENDAHL, F. PARMIGIANI, P. PETROPOULOS, B. RESAN, K. WEINGARTEN, T. ELLERMEYER, J. LUTZ, M. MÖLLER, M. HÜBNER, BECKER J., C. KOOS, W. FREUDE und J. LEUTHOLD: *Single Source Optical OFDM Transmitter and Optical FFT Receiver Demonstrated at Line Rates of 5.4 and 10.8 Tbit/s*. In: *Optical Fiber Communication Conference and Exposition (OFC) and the National Fiber Optic Engineers Conference (NFOEC)*. Optical Society of America, 2010.

[HSS+11] HILLERKUSS, D., R. SCHMOGROW, T. SCHELLINGER, M. JORDAN, M. WINTER, G. HUBER, T. VALLAITIS, R. BONK, F. KLEINOW, F. FREY, M. ROEGER, S. KOENIG, A. LUDWIG, A. MARCULESCU, J. LI, M. HOH, M. DRESCHMANN, J. MEYER, S. BEN EZRA, N. NARKISS, B. NEBENDAHL, F. PARMIGIANI, P. PETROPOULOS, B. RESAN, A. OEHLER, K. WEINGARTEN, T. ELLERMEYER, J. LUTZ, M. MOELLER, M. HUEBNER, J. BECKER, C. KOOS, W. FREUDE und J. LEUTHOLD: *26 Tbit/s line-rate super-channel transmission utilizing all-optical fast Fourier transform processing*. Nature Photonics, 5(6), June 2011.

[LWF+10] LEUTHOLD, J., M. WINTER, W. FREUDE, C. KOOS, D. HILLERKUSS, T. SCHELLINGER, R. SCHMOGROW, T. VALLAITIS, R. BONK, A. MARCULESCU, J. LI, M. DRESCHMANN, J. MEYER, M. HUEBNER, J. BECKER, S. BEN EZRA, N. NARKISS, B. NEBENDAHL, F. PARMIGIANI, P. PETROPOULOS, B. RESAN, A. OEHLER, K. WEINGARTEN, T. ELLERMEYER, J. LUTZ und M. MÖLLER: *All-Optical FTT Signal Processing of a 10.8 Tb/s Single Channel OFDM Signal*. In: *Integrated Photonics Research, Silicon and Nanophotonics (IPR) and Photonics in Switching (PS)*. Optical Society of America, 2010.

[MDK+13] MEYER, J., M. DRESCHMANN, D. KARNICK, P. C. SCHINDLER, W. FREUDE, J. LEUTHOLD und BECKER J.: *A novel system on chip for software-defined high-speed OFDM signal processing*. In: *26th Symposium on Integrated Circuits and Systems Design (SBCCI)*, September 2013.

[MHB+11] MEYER, J., M. HÜBNER, L. BRAUN, O. SANDER, J. NOGUERA, R. STEWART und J. BECKER: *FPGA Startup Through Sequential Partial and Dynamic Reconfiguration*. In: VOROS, NIKOLAOS, AMAR MUKHERJEE, NICOLAS SKLAVOS, KONSTANTINOS MASSELOS und MICHAEL HUEBNER (Herausgeber): *VLSI 2010 Annual Symposium*, Band 105 der Reihe *Lecture Notes in Electrical Engineering*. Springer Netherlands, 2011.

[MMD+12] MEYER, J., S. MENZEL, M. DRESCHMANN, R. SCHMOGROW, D. HILLERKUSS, W. FREUDE, J. LEUTHOLD und J. BECKER: *Ultra High Speed Digital Down Converter Design for Virtex-6 FPGAs*. In: *Proceedings of the 17th International OFDM Workshop (InOWo'12)*, 2012.

[MNH+11a] MEYER, J., J. NOGUERA, M. HÜBNER, L. BRAUN, O. SANDER, R.M. GIL, R. STEWART und J. BECKER: *Fast Start-up for Spartan-6 FPGAs using Dynamic Partial Reconfiguration*. In: *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2011.

[MNH+11b] MEYER, J., J. NOGUERA, M. HUEBNER, R. STEWART und J. BECKER: *Embedded Systems Start-Up under Timing Constraints on Modern FPGAs*. In: *International Conference on Field Programmable Logic and Applications (FPL)*, 2011.

[MNH+13] MEYER, J., J. NOGUERA, M. HÜBNER, R. STEWART und J. BECKER: *Embedded Systems Start-Up Under Timing Constraints on Modern FPGAs*. In: ATHANAS, PETER, DIONISIOS PNEVMATIKATOS und NICOLAS SKLAVOS (Herausgeber): *Embedded Systems Design with FPGAs*. Springer New York, 2013.

[MNS+11] MEYER, J., J. NOGUERA, R. STEWART, M. HÜBNER und J. BECKER: *Fast Startup for Xilinx FPGAs*. Xcell, 75, 2011.

[MSA+14] MEDER, L., P. SCHINDLER, A. AGMON, M. MELTSIN, R. BONK, J. MEYER, M. DRESCHMANN, A. TOLMACHEV, R. HILGENDORF, M. NAZARATHY, S. BEN-EZRA, PFEIFFER T., W. FREUDE, J. LEUTHOLD, C. KOOS und J. BECKER: *Flexible Real-Time Transmitter at 10Gbit/s for SCFDMA PONs Focusing on Low-Cost ONUs*. In: *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, October 2014.

[RMR+12] ROTH, C., J. MEYER, M. RÜCKAUER, O. SANDER und J. BECKER:

Efficient execution of networked MPSoC models by exploiting multiple platform levels. International Journal of Reconfigurable Computing, January 2012.

[RMS+12] RUECKAUER, M., J. MEYER, T. SCHUBERT, M. HUBNER, D. SCHEURER und J. BECKER: *Realtime PCI Express monitoring for self adaptive reconfigurable systems*. In: *9th International Multi-Conference on Systems, Signals and Devices (SSD)*, 2012.

[SAW+14] SCHINDLER, P.C., A. AGMON, S. WOLF, R. BONK, L. MEDER, M. MELTSIN, A. LUDWIG, R. SCHMOGROW, M. DRESCHMANN, J. MEYER, J. BECKER, M. NAZARATHY, S. BEN-EZRA, T. PFEIFFER, W. FREUDE, J. LEUTHOLD und C. KOOS: *Ultra-Dense, Single-Wavelength DFT-Spread OFDMA PON with Laserless 1.2 Gb/s ONU Ready for Silicon Photonics Integration*. Journal of Lightwave Technology, PP(99), December 2014.

[SBP+13] SCHMUCK, H., R. BONK, W. POEHLMANN, C. HASLACH, W. KUEBART, D. KARNICK, J. MEYER, D. FRITZSCHE, E. WEIS, J. BECKER, W. FREUDE und PFEIFFER T.: *Demonstration of SOA-assisted Open Metro-Access Infrastructure for Heterogeneous Services*. In: *39th European Conference and Exhibition on Optical Communication (ECOC)*, September 2013.

[SBP+14] SCHMUCK, H., R. BONK, W. POEHLMANN, C. HASLACH, W. KUEBART, D. KARNICK, J. MEYER, D. FRITZSCHE, E. WEIS, J. BECKER, W. FREUDE und T. PFEIFFER: *Demonstration of an SOA-assisted open metro-access infrastructure for heterogeneous services*. Optics Express, 22(1), January 2014.

[SHD+10] SCHMOGROW, R., D. HILLERKUSS, M. DRESCHMANN, M. HUEBNER, M. WINTER, J. MEYER, B. NEBENDAHL, C. KOOS, J. BECKER, W. FREUDE und J. LEUTHOLD: *Real-Time Software-Defined Multiformat Transmitter Generating 64QAM at 28 GBd*. Photonics Technology Letters, IEEE, 22(21), 2010.

[SMW+12] SCHMOGROW, R., M. MEYER, S. WOLF, B. NEBENDAHL, D. HILLERKUSS, B. BAEUERLE, M. DRESCHMANN, J. MEYER, M. HUEBNER, J. BECKER, C. KOOS, W. FREUDE und J. LEUTHOLD: *150 Gbit/s Real-Time Nyquist Pulse Transmission Over 150 km SSMF Enhanced by DSP with Dynamic Precision*. In: *Optical Fiber Communication Conference and Exposition (OFC) and the National Fiber Optic Engineers Conference (NFOEC)*. Optical Society of America, 2012.

[SNW+12] SCHMOGROW, R., B. NEBENDAHL, M. WINTER, A. JOSTEN, D. HILLERKUSS, S. KOENIG, J. MEYER, M. DRESCHMANN, M. HUEBNER,

C. KOOS, J. BECKER, W. FREUDE und J. LEUTHOLD: *Error Vector Magnitude as a Performance Measure for Advanced Modulation Formats, Corrections in: IEEE Photon. Technol. Lett. 24 (2012) 2198*. IEEE Photonics Technology Letters, 24(1), 2012.

[SSD+13a] SCHINDLER, P. C., R. SCHMOGROW, M. DRESCHMANN, J. MEYER, D. HILLERKUSS, I. TOMKOS, J. PRAT, H. G. KRIMMEL, T. PFEIFFER, P. KOURTESSIS, J. BECKER, C. KOOS, W. FREUDE und J. LEUTHOLD: *Flexible WDM-PON with Nyquist-FDM and 31.25 Gbit/s per Wavelength Channel Using Colorless, Low-Speed ONUs*. In: *Optical Fiber Communication Conference and Exposition (OFC) and the National Fiber Optic Engineers Conference (NFOEC)*. Optical Society of America, 2013.

[SSD+13b] SCHINDLER, P. C., R. SCHMOGROW, M. DRESCHMANN, J. MEYER, I. TOMKOS, J. PRAT, H.-G. KRIMMEL, T. PFEIFFER, P. KOURTESSIS, A. LUDWIG, D. KARNICK, D. HILLERKUSS, J. BECKER, C. KOOS, W. FREUDE und J. LEUTHOLD: *Colorless FDMA-PON With Flexible Bandwidth Allocation and Colorless, Low-Speed ONUs (Invited)*. Journal of Optical Communications and Networking, 5(10), October 2013.

[SWH+11] SCHMOGROW, R., M. WINTER, D. HILLERKUSS, B. NEBENDAHL, S. BEN-EZRA, J. MEYER, M. DRESCHMANN, M. HUEBNER, J. BECKER, C. KOOS, W. FREUDE und J. LEUTHOLD: *Real-time OFDM transmitter beyond 100 Gbit/s*. Optics Express, 19(13), June 2011.

[SWM+11] SCHMOGROW, R., M. WINTER, M. MEYER, D. HILLERKUSS, B. NEBENDAHL, J. MEYER, M. DRESCHMANN, M. HUEBNER, J. BECKER, C. KOOS, W. FREUDE und J. LEUTHOLD: *Real-Time Nyquist Pulse Modulation Transmitter Generating Rectangular Shaped Spectra of 112 Gbit/s 16QAM Signals*. In: *Advanced Photonics*. Optical Society of America, 2011.

[SWM+12] SCHMOGROW, R., M. WINTER, M. MEYER, D. HILLERKUSS, S. WOLF, B. BAEUERLE, A. LUDWIG, B. NEBENDAHL, S. BEN-EZRA, J. MEYER, M. DRESCHMANN, M. HUEBNER, J. BECKER, C. KOOS, W. FREUDE und J. LEUTHOLD: *Real-time Nyquist pulse generation beyond 100 Gbit/s and its relation to OFDM*. Optics Express, 20(1), Jan 2012.

[SWN+11] SCHMOGROW, R., M. WINTER, B. NEBENDAHL, D. HILLERKUSS, J. MEYER, M. DRESCHMANN, M. HUEBNER, J. BECKER, C. KOOS, W. FREUDE und J. LEUTHOLD: *101.5 Gbit/s Real-Time OFDM Transmitter with 16QAM Modulated Subcarriers*. In: *Optical Fiber Communication Conference and Exposition (OFC) and the National Fiber Optic Engineers Conference (NFOEC)*. Optical Society of America, 2011.

Patente

[SHNS+11] STEWART, RODNEY, MICHAEL HUEBNER, JUAN J. NOGUERA SERRA, ROBERT P. ESSER, JÜRGEN BECKER, OLIVER SANDER, MATTHIAS TRAUB und JOACHIM H. MEYER: *US 7932743 B1: Sequentially configurable programmable integrated circuit*, 2011.

Joachim Meyer

HW/SW Co-Design Framework für Hochgeschwindigkeits-OFDM Signalverarbeitung

Der Bedarf an Kommunikation in unserer Gesellschaft steigt stetig, den größten Zuwachs verzeichnet hierbei derzeit die drahtlose Signalübertragung. Dabei werden immer komplexere Verfahren zur Signalverarbeitung entwickelt um die Datenraten oder die Übertragungsqualität zu steigern. Eines dieser Verfahren, welches sich im drahtlosen Bereich etabliert hat, stellt das orthogonale Frequenzmultiplexverfahren (OFDM) dar.
Im Rahmen dieser Arbeit wurde ein HW/SW Co-Design Framework entwickelt, welche es ermöglicht, auf einfachste Weise, angepasste Multiprozessor System-on-Chips (MPSOC) zu erstellen. Durch Einsatz dieses Frameworks lassen sich für moderne OFDM-Systeme neue Kompromisse zwischen Leistungsfähigkeit und Flexibilität erzielen.
Die Möglichkeiten des Frameworks wurden im Rahmen der Arbeit durch aufgebaute Prototypen mit der Hilfe von rekonfigurierbarer Hardware, sogenannten Field Programmable Gate Arrays (FPGAs), demonstriert. Anhand unterschiedlicher Experimente zur Hochgeschwindigkeits-OFDM Übertragung mit Echtzeit-signalverarbeitung wurde nicht nur die Funktionalität der Systeme nachgewiesen, es wurden auch Datenraten im Gb/s-Bereich erzielt, was bisher lediglich unflexiblen, dedizierten Schaltkreisen vorbehalten war.