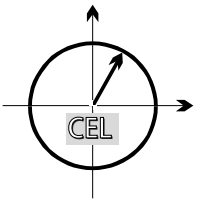
Forschungsberichte aus dem
Institut für Nachrichtentechnik des
Karlsruher Instituts für Technologie



Michael Schwall

Turbo-Entzerrung: Implementierungsaspekte für Software Defined Radios

Band 33

Druck: Frick Kreativbüro & Onlinedruckerei e.K.
Brühlstraße 6
86381 Krumbach

ISSN: 1433-3821

Forschungsberichte aus dem Institut für Nachrichtentechnik des Karlsruher Instituts für Technologie

Herausgeber: Univ.-Prof. Dr. rer. nat. Friedrich K. Jondral

Band 1	Marcel Kohl	**Simulationsmodelle für die Bewertung von Satellitenübertragungsstrecken im 20/30 GHz Bereich**
Band 2	Christoph Delfs	**Zeit-Frequenz-Signalanalyse: Lineare und quadratische Verfahren sowie vergleichende Untersuchungen zur Klassifikation von Klaviertönen**
Band 3	Gunnar Wetzker	**Maximum-Likelihood Akquisition von Direct Sequence Spread-Spectrum Signalen**
Band 4	Anne Wiesler	**Parametergesteuertes Software Radio für Mobilfunksysteme**
Band 5	Karl Lütjen	**Systeme und Verfahren für strukturelle Musteranalysen mit Produktionsnetzen**
Band 6	Ralf Machauer	**Multicode-Detektion im UMTS**
Band 7	Gunther M. A. Sessler	**Schnell konvergierender Polynomial Expansion Multiuser Detektor mit niedriger Komplexität**
Band 8	Henrik Schober	**Breitbandige OFDM Funkübertragung bei hohen Teilnehmergeschwindigkeiten**
Band 9	Arnd-Ragnar Rhiemeier	**Modulares Software Defined Radio**
Band 10	Mustafa Mengüç Öner	**Air Interface Identification for Software Radio Systems**

Forschungsberichte aus dem Institut für Nachrichtentechnik des Karlsruher Instituts für Technologie
Herausgeber: Univ.-Prof. Dr. rer. nat. Friedrich K. Jondral

Band 11	Fatih Çapar	**Dynamische Spektrumverwaltung und elektronische Echtzeitvermarktung von Funkspektren in Hotspotnetzen**
Band 12	Ihan Martoyo	**Frequency Domain Equalization in CDMA Detection**
Band 13	Timo Weiß	**OFDM-basiertes Spectrum Pooling**
Band 14	Wojciech Kuropatwiński-Kaiser	**MIMO-Demonstrator basierend auf GSM-Komponenten**
Band 15	Piotr Rykaczewski	**Quadraturempfänger für Software Defined Radios: Kompensation von Gleichlauffehlern**
Band 16	Michael Eisenacher	**Optimierung von Ultra-Wideband-Signalen (UWB)**
Band 17	Clemens Klöck	**Auction-based Medium Access Control**
Band 18	Martin Henkel	**Architektur eines DRM-Empfängers und Basisbandalgorithmen zur Frequenzakquisition und Kanalschätzung**
Band 19	Stefan Edinger	**Mehrträgerverfahren mit dynamisch-adaptiver Modulation zur unterbrechungsfreien Datenübertragung in Störfällen**
Band 20	Volker Blaschke	**Multiband Cognitive Radio-Systeme**

Forschungsberichte aus dem Institut für Nachrichtentechnik des Karlsruher Instituts für Technologie
Herausgeber: Univ.-Prof. Dr. rer. nat. Friedrich K. Jondral

Band 21 Ulrich Berthold
Dynamic Spectrum Access using OFDM-based Overlay Systems

Band 22 Sinja Brandes
Suppression of Mutual Interference in OFDM-based Overlay Systems

Band 23 Christian Körner
Cognitive Radio – Kanalsegmentierung und Schätzung von Periodizitäten

Band 24 Tobias Renk
Cooperative Communications: Network Design and Incremental Relaying

Band 25 Dennis Burgkhardt
Dynamische Reallokation von spektralen Ressourcen in einem hierarchischen Auktionssystem

Band 26 Stefan Nagel
Portable Waveform Development for Software Defined Radios

Band 27 Hanns-Ulrich Dehner
Interferenzuntersuchungen für inkohärente Multiband Ultra-Breitband (UWB) Übertragung

Band 28 Maximilian Hauske
Signalverarbeitung für optoelektronische Sensoren

Band 29 Jens Elsner
Interference Mitigation in Frequency Hopping Ad Hoc Networks

Band 30 Georg Vallant
Modellbasierte Entzerrung von Analog/Digital-Wandler-Systemen

Forschungsberichte aus dem Institut für Nachrichtentechnik des Karlsruher Instituts für Technologie
Herausgeber: Univ.-Prof. Dr. rer. nat. Friedrich K. Jondral

Band 31 Martin Braun
OFDM Radar Algorithms in Mobile Communication Networks

Band 32 Michael Mühlhaus
Automatische Modulationsartenerkennung in MIMO-Systemen

Band 33 Michael Schwall
Turbo-Entzerrung: Implementierungsaspekte für Software Defined Radios

Vorwort des Herausgebers

Das Software Defined Radio (SDR) Prinzip sagt aus, dass die Funktionalitäten von Sender und Empfänger eines solchen Funkgeräts ausschließlich von der verwendeten Software bestimmt werden. In einem SDR nutzt die Software (in diesem Zusammenhang auch als Implementierung eines Standards oder einer Wellenform zu interpretieren) eine geeignete (Hardware-) Plattform. Die Plattform ist ihrerseits aus der Funkeinheit, die das an der Antenne anliegende Empfangssignal in seine (komplexe) Basisbanddarstellung bzw. das Sendesignal aus der Basisbanddarstellung in ein (reelles) Bandpasssignal transformiert, und der digitalen Signalverarbeitungseinheit, die die Basisbandsignalverarbeitung und die Funktionen der höheren Protokollschichten ausführt, aufgebaut. Ein schönes Beispiel für eine Plattform bietet das, auch am Institut für Nachrichtentechnik des KIT in Forschung und Lehre verwendete, USRP[1], eine spezielle Hardware, die unter anderem ein Field Programmable Gate Array (FPGA) sowie digitale Signalprozessoren (DSPs) umfasst, im Zusammenspiel mit einem General Purpose Processor (GPP). Ein anderes, wesentlich leistungsfähigeres, aber damit auch komplexeres Beispiel liefert die Streitkräftegemeinsame Verbundfähige Funkgeräte Ausstattung (SVFuA) der Bundeswehr [TEE10].

Im Gegensatz zu Endgeräten im öffentlichen Netz (z.B. Smart Phones) greifen SDRs nicht auf rechenleistungs- und stromverbrauchsoptimierte Application Specific Integrated Circuits (ASICs) zurück, da diese monolithisch integrierte, fest verdrahtete Schaltungen sind, deren An-

[1] Universal Software Radio Peripheral

wendung dem Prinzip der flexiblen Programmierbarkeit und damit dem SDR Paradigma widerspricht.

Das Ziel einer jeden digitalen Funkübertragung besteht darin, Bits von einem Ort zum anderen fehlerfrei zu übertragen. Nun ist die Signalübertragung widrigen Umwelteinflüssen wie Brechung, Beugung, Reflexion, Dopplereffekt und vor allem der Mehrwegeausbreitung ausgesetzt. Da sich eigentlich nur für den Fall einer Störung des Empfangssignals durch additives weißes Gaußsches Rauschen (Additive White Gaussian Noise, AWGN) optimale Demodulatoren angeben lassen, besteht ein großer Teil des SDR Empfängers aus Algorithmen, die das Empfangssignal so umformen, dass es am Symbolentscheider als ein allein durch AWGN gestörtes Nutzsignal erscheint. Besondere Bedeutung kommt dabei dem Entzerrer zu, der empfängerseitig der kohärenten additiven Überlagerung der Mehrwegesignale dient. Die zur Kanalentzerrung eingesetzten Methoden werden mit wachsender Signalbandbreite komplizierter und aufwändiger, ein Effekt, der z.B. für Long Term Evolution (LTE) Signale eine nicht zu unterschätzende Rolle spielt.

Als Mittel zur effektiven Behandlung der Mehrwegausbreitung hat sich die Turbo-Entzerrung herausgestellt, die den senderseitig aufgeprägten Fehlerschutz im Empfänger zur Entzerrung nutzt. In mobilen Endgeräten öffentlicher Mobilfunknetze werden für die Entzerrung ASICs eingesetzt. Das ist in SDRs nicht möglich, weshalb für sie bei Implementierung der Turbo-Entzerrung die Parallelisierbarkeit und Festkommaaspekte besonders zu berücksichtigen sind. Hier setzt die vorliegende Dissertation von Michael Schwall an. Sie liefert allgemeingültige Optimierungsansätze bei der (Software) Implementierung von Turbo-Entzerrern, die auf ein breites Spektrum von SDR Plattformen anwendbar sind.

Karlsruhe, im Dezember 2014

Friedrich Jondral

Turbo-Entzerrung: Implementierungsaspekte für Software Defined Radios

Zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEURS

von der Fakultät für
Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie

genehmigte

DISSERTATION

von

Dipl.-Ing. Michael Schwall

geb. in Wittlich

Tag der mündlichen Prüfung: 27. November 2014
Hauptreferent: Univ.-Prof. Dr. rer. nat. Friedrich K. Jondral
Korreferent: Prof. Dr.-Ing. Werner Henkel

Inhaltsverzeichnis

KAPITEL 1

Einleitung

Die zuverlässige Übertragung von Bits ist das primäre Ziel eines *digitalen* Funksystems. Die Herausforderungen dieser Funkübertragung von einem Sender zu einem Empfänger ergeben sich bei der Kompensation der physikalischen Effekte, die im Funkkanal auf das Funksignal wirken. Abhängig von der Datenrate der Übertragung führen die Ausbreitungsbedingungen dazu, dass sich unterschiedliche Kopien des Funksignals zeitverzögert, phasenverschoben und gedämpft am Empfänger überlagern und Interferenz entstehen kann. Zur Rekonstruktion des gesendeten Signals ist dann empfängerseitig eine *Entzerrung* erforderlich.

Während der Einfluss der physikalischen Effekte auf das Funksignal im Wesentlichen durch die Topografie, die Morphografie und die Bebauung der Umgebung vorgegeben ist, wird die Entzerrung und deren Komplexität aus Sicht der digitalen Signalverarbeitung zusätzlich durch die gegenwärtigen Entwicklungen in der Nachrichtentechnik beeinflusst: Die Einführung *breitbandiger* und *netzwerkfähiger* Funkstandards, wie z.B. Long Term Evolution (LTE) [LTE14] oder Coalition Wideband Networking Waveform [WNW14], die hochdatenratige Multimediadienste ermöglichen und die Mobilität des Funksystems erhöhen, führt zu einer Verstärkung des Interferenzproblems. Zur Erfüllung der hohen technischen Anforderungen sind dementsprechend *rechenintensive* Verfahren notwendig, die eine zuverlässige Rekonstruktion

des Funksignals sicherstellen. Im Mobilfunk wird dazu der enorme Rechenaufwand auf anwendungsspezifische integrierte Schaltungen (*engl.* Application-Specific Integrated Circuit, ASIC) in den mobilen Endgeräten, wie z.B. in einem Smartphone, ausgelagert.

Die in dieser Arbeit betrachtete *Turbo-Entzerrung* berücksichtigt erstmals den senderseitig aufgeprägten Fehlerschutz im Funksignal zur Entzerrung des Empfangssignals und ermöglicht im Vergleich zu konventionellen Verfahren dadurch eine zuverlässigere Rekonstruktion. Der Ansatz basiert auf einer Veröffentlichung von Douillard *et al.* [DJB+95] aus dem Jahr 1995 und geht auf das Prinzip der Turbo-Codes von Berrou *et al.* [BGT93] zurück. Die verbesserte Leistungsfähigkeit des iterativen Verfahrens erfordert allerdings komplexe Algorithmen und verursacht infolge dessen einen deutlichen Mehraufwand bei der digitalen Signalverarbeitung.

Seit der ersten Publikation von Rohde [Roh85] aus dem Jahr 1985 hat sich das Konzept des *Software Defined Radios* (SDR) im militärischen und industriellen Umfeld sowie in der Forschung etabliert. Die ursprüngliche Idee eines Funkgeräts, dessen Funktionalität ausschließlich durch Software bestimmt wird, ist mittlerweile durch Kauflösungen verschiedener Hersteller, wie z.B. Ettus Research [Ett14] oder Nutaq [Nut14], realisiert. Die erhältlichen Plattformen unterscheiden sich unter anderem in der Architektur, den Abmessungen sowie in der Mobilität und der verfügbaren Rechenleistung. Die Rekonfigurierbarkeit eines SDRs und die damit einhergehenden Vorteile wie Multistandardfähige oder wiederverwendbare Funksysteme setzen allerdings voraus, dass die integrierte Signalverarbeitungshardware programmiert bzw. beschrieben werden kann.

Aufgabenstellung

Im Vergleich zu einem ASIC, der für eine dedizierte Aufgabe optimiert und in hohen Stückzahlen produziert wird, erfolgt die digitale Signalverarbeitung in einem SDR auf generischen Signalverarbeitungseinheiten, z.B. auf einem General Purpose Processor oder Field Programmable Gate Array. Obwohl dies aus Sicht der Rekonfigurierbarkeit eine sehr flexible Lösung darstellt und letztlich den Vorteil

eines Software-definierten Funkgeräts begründet, ist jedoch aufgrund dessen die Rechenleistung geringer und die elektrische Leistungsaufnahme höher als bei einem monolithisch integrierten ASIC.

Um dennoch zu erreichen, dass zukünftig breitbandige und mobile Funkkommunikation mit SDR-basierten Funksystemen zuverlässig mithilfe eines Turbo-Entzerrers in Echtzeit durchgeführt werden kann, muss dessen Implementierung an die Signalverarbeitungshardware eines SDRs angepasst und entsprechend optimiert werden. Da dies aufgrund der komplexen Algorithmen und des iterativen Ablaufs des Verfahrens eine nicht-triviale Aufgabe darstellt, befasst sich die vorliegende Arbeit mit den *Implementierungsaspekten* der Turbo-Entzerrung für Software Defined Radios. Im Gegensatz zu Chip-spezifischen Lösungen werden dazu die *Parallelisierungs-* und *Festkommaaspekte* im Detail analysiert. Da sich die integrierte Signalverarbeitungshardware zwischen verschiedenen SDRs teils stark unterscheidet, stellen die Ergebnisse dieser Analysen *allgemeingültige* Optimierungsmöglichkeiten dar und können auf ein breites Spektrum von Prozessoren oder Logiken angewendet werden.

Gliederung der Arbeit

Nach einer Einführung in Software Defined Radio liegt der Fokus des Kapitels 2 auf der digitalen Signalverarbeitungshardware. Die verschiedenen Prozessoren und Logiken, die sich für den Einsatz in dieser Technologie qualifizieren und bereits integriert sind, werden vorgestellt und wesentliche Merkmale hervorgehoben. Themen wie Mehrkernprozessoren oder Festkommaarithmetik bilden dabei die technischen Grundlagen für die in dieser Arbeit untersuchten Implementierungsaspekte.

Kapitel 3 beschäftigt sich mit den Grundlagen der digitalen Funkübertragung und stellt das verwendete Systemmodell vor. Zunächst wird dazu eine mathematische Darstellung des Funkkanals zur Modellierung der physikalischen Effekte eingeführt. Anschließend folgt die Beschreibung des Senders, der nach einer Bit-Interleaved Coded Modulation arbeitet und dessen Prinzip in vielen modernen Funkstandards, wie z.B. LTE, wiederzufinden ist. Den Abschluss des Kapitels bildet

die Diskussion des optimalen Empfängers für die gegebene Funkübertragung. Obwohl eine Implementierung dieses Empfängers aufgrund der immensen Rechenkomplexität nicht möglich ist, legt der Ansatz jedoch den Grundstein zur Einführung der Turbo-Entzerrung.

Die detaillierte Beschreibung der Turbo-Entzerrung ist Gegenstand von Kapitel 4. Nach der Erläuterung des grundsätzlichen Funktionsprinzips werden die beiden Kernkomponenten des iterativen Verfahrens vorgestellt: Ein Soft-Input Soft-Output (SISO) Entzerrer und ein SISO-Decoder. Der Entzerrer wird in dieser Arbeit mittels eines klassischen MMSE[1]-basierten Frequenzbereichsentzerrers realisiert, der mithilfe der Soft Interference Cancellation Methode in den Turbo-Entzerrer integriert wird. Der SISO-Decoder wird durch einen Max*-Log-MAP Decoder, eine recheneffiziente Modifikation des Maximum *a posteriori* (MAP) Algorithmus, umgesetzt. Die Darstellung des Konvergenzverhaltens mithilfe von Extrinsic Information Transfer (EXIT) Diagrammen ermöglicht eine effiziente Bewertung der Leistungsfähigkeit der Turbo-Entzerrung und wird am Ende des Kapitels erläutert.

Die Analyse von Festkommaaspekten in Kapitel 5 ist ein wesentlicher Schritt zur Implementierung der Turbo-Entzerrung in Festkommaarithmetik. Infolge der limitierten Dynamik bei der Darstellung von Zahlenwerten wird die Konvergenz des iterativen Verfahrens beeinträchtigt und die erreichbare Zuverlässigkeit bei der Rekonstruktion des Funksignals reduziert. Mithilfe analytischer Modelle und Simulationen werden der Einfluss der Arithmetik auf die Kernkomponenten untersucht und quantitative Abhängigkeiten identifiziert. Die Zusammenhänge ermöglichen anschließend die Dimensionierung effizienter Festkommadarstellungen zur Umsetzung der Turbo-Entzerrung. Die theoretischen Ergebnisse werden durch Referenzimplementierungen auf Prozessoren und Logiken verifiziert.

Kapitel 6 widmet sich der Parallelisierbarkeit der Turbo-Entzerrung. Nach der Erläuterung grundsätzlicher Prinzipien und Grenzen der Parallelisierung konzentrieren sich die Analysen zunächst auf die Kernkomponenten des Verfahrens. Auf unterschiedlichen Ebenen werden die Algorithmen der SISO-Entzerrung und -Decodierung hinsichtlich

[1] Minimum Mean Square Error

Nebenläufigkeiten untersucht und der theoretische Parallelisierungsgewinn ermittelt. Die praktische Evaluation wird erneut durch Referenzimplementierungen auf Prozessoren durchgeführt. Den Abschluss des Kapitels bildet die Parallelisierung der gesamten Turbo-Entzerrung mithilfe des aus der Informationstechnik stammenden Pipelinings. Die Verifikation dieses Ansatzes geschieht erneut durch Implementierungen.

Die Zusammenfassung in Kapitel 7 hebt nochmal die wichtigsten Ergebnisse der Arbeit hervor und gibt einen Ausblick für mögliche anschließende Arbeiten.

KAPITEL 2

Software Defined Radio

Die Vision eines Funkgeräts, dessen Funktionalität vollständig durch Software vorgegeben wird, geht auf eine Veröffentlichung von Rohde aus dem Jahre 1985 zurück [Roh85]. Die treffende und mittlerweile etablierte Bezeichnung *Software Radio* wurde erstmals von Mitola in einem Beitrag von 1992 geprägt [Mit92]. Ein Software Radio verarbeitet Funksignale digital und führt im Falle eines Empfängers die Digitalisierung bereits am Antenneneingang durch. In Senderichtung wird gleichermaßen das generierte digitale Funksignal erst unmittelbar vor der Antenne analog gewandelt. Die digitale Signalverarbeitung findet auf programmierbaren Prozessoren oder beschreibbaren Logiken statt.

Der Vorteil dieses Ansatzes resultiert aus der Abbildung der Funktionalität eines Funkgeräts, in diesem Kontext als *Wellenform* bezeichnet, auf immaterielle Software und die somit erzielte Trennung vom Gerät selbst, der sogenannten *Plattform*. Die Software kann dabei beliebig oft vervielfältigt, ausgetauscht oder modifiziert werden und ist nicht an eine Signalverarbeitungseinheit gebunden. Wellenformen sind folglich portabel und aktualisierbar, während Plattformen gleichzeitig interoperabel sind. Die technischen Probleme dieses idealen Ansatzes entstehen bei der Realisierung der Analog-zu-Digital-Wandlung. Die vollständige Digitalisierung des für die Funkkommunikation relevanten elektromagnetischen Spektrums am Antenneneingang erfordert Wandler mit Abtastraten im zweistelligen Gigahertz-Bereich. Nach

dem heutigen Stand der Elektronik sind im Konsumerbereich Wandlerraten von maximal 250 MHz bei einer Auflösung von 14-bit möglich [ADC14].

Ein *Software Defined Radio* (SDR) stellt eine technisch realisierbare Variante des idealen Ansatzes dar. Das Einfügen einer Hochfrequenzstufe, die eine Anpassung der Funksignale in einem beobachteten Teilband des Spektrums an den Arbeitsbereich der Analog-zu-Digital-Wandler durchführt, ermöglicht eine Reduktion der erforderlichen Wandlerraten, jedoch auf Kosten der instantanen Beobachtungsbandbreite. Rauscharme Verstärker, analoge Filter sowie kaskadierte Hochfrequenzmischer sind die wesentlichen Komponenten dieser Hochfrequenztechnik (HF). Der schematische Aufbau eines SDRs ist in Abbildung 2.1 dargestellt.

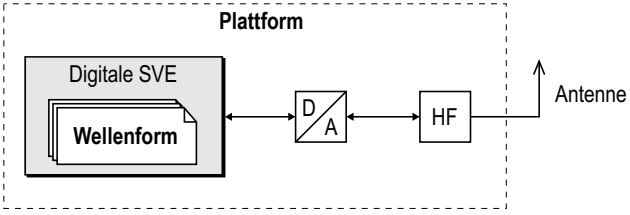


Abb. 2.1: Schematischer Aufbau eines SDRs

Aufgrund der Möglichkeit mehrere Funksysteme in einem Funkgerät zu vereinen und ebenso für zukünftige Funktechniken eine flexible Lösung zu bieten, hat die SDR-Technologie in den vergangenen Jahren Einzug in verschiedene militärische und zivile Bereiche gehalten. Ein Beispiel ist die streitkräftegemeinsame, verbundfähige Funkgeräteausstattung (SVFuA) [TEE10] die seit kurzem die SDR-basierte Lösung für die multilaterale Kommunikationsinfrastrukturen der Bundeswehr darstellt und damit als zentrale Komponente der vernetzten Operationsführung gilt.

Die Rekonfigurierbarkeit eines Funkgeräts führt allerdings zu Herausforderungen bei der Implementierung von breitbandigen Wellenformen, die hohe Anforderungen an die Rechenleistung stellen. Im Vergleich zu applikationsspezifischen Funksystemen, die zur digitalen Signalverarbeitung hochintegrierte und optimierte Schaltungen verwen-

den, ist die Rechenkapazität eines SDRs aufgrund der Programmier- und Beschreibbarkeit stark limitiert. Es gilt daher bereits bei der Umsetzung eine Optimierung der Software zur effizienten Nutzung der Rechenkapazität durchzuführen. Im Fokus dieses Kapitels steht daher die *digitale Signalverarbeitungseinheit* eines SDRs. Es werden im folgenden Abschnitt 2.1 die verschiedenen Prozessoren und Logiken, die sich für den Einsatz in dieser Technologie qualifizieren und bereits verwendet werden, vorgestellt und wesentliche Unterschiede erläutert. Aufbauend auf diesen Eigenschaften und Merkmalen beschäftigen sich Kapitel 5 und Kapitel 6 mit der Optimierung der Turbo-Entzerrung. Weiterführende Literatur zum Themengebiet SDR findet sich in [JMW02], [Ree02] und [WP13].

2.1 Digitale Signalverarbeitungseinheit

Die digitale Signalverarbeitungseinheit (SVE) ist das Kernstück eines SDRs. Sie stellt die Rechenkapazitäten zur Verfügung, um Wellenformen auf der Plattform implementieren zu können. Die eingesetzte Technologie ist vorwiegend von den Rahmenbedingungen des Einsatzes und von der Energieversorgung des Gerätes abhängig. Je nach Anforderung sind zudem verschiedene Technologien zur kooperativen Signalverarbeitung auf einer sogenannten heterogenen Plattform integriert. Abbildung 2.2 zeigt exemplarisch eine heterogene SDR-Plattform bestehend aus einem Ettus USRP[1] B210 [Ett14] und einem Notebook. Die digitale SVE bilden ein Field Programmable Gate Array und ein General Purpose Prozessor. Der Datenaustausch geschieht über USB (Universal Serial Bus) 3.0.

Die Komponenten der SVE lassen sich aus Sicht des Schaltungsaufbaus in die zwei zentralen Klassen *Prozessoren* und *Logiken* unterteilen. Während ein Prozessor über ein Rechen- und Steuerwerk verfügt und einen vorgegeben Programmcode ausführt, besteht eine Logik aus einer Vielzahl elementarer Logikgatter. Eine Funktionalität wird der Logik durch das gezielte Verknüpfen und Parametrisieren der Gatter

[1] Universal Software Radio Peripheral

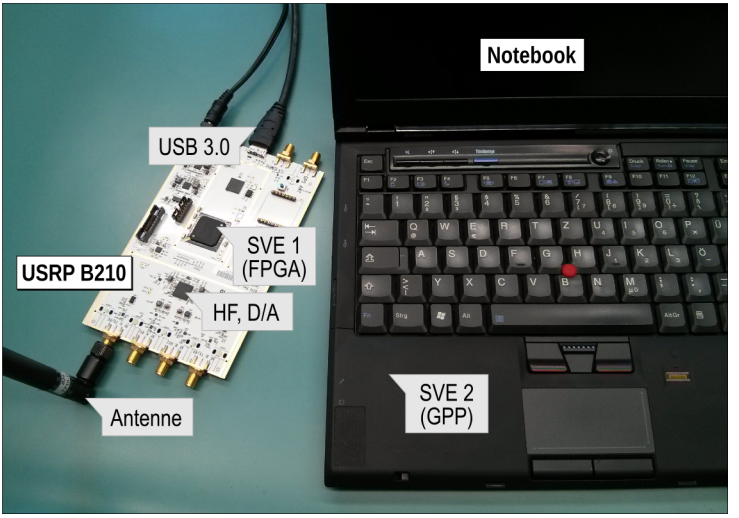


Abb. 2.2: Heterogene SDR-Plattform

aufgeprägt. Im Fall der Prozessoren spricht man aufgrund der Individualität von *Programmierung*, im letzteren von *Beschreibung*.

2.1.1 Prozessor

Prozessoren verfügen im Gegensatz zu Logiken über ein Rechenwerk, die sogenannte Arithmetisch-logische Einheit (*engl.* Arithmetic Logic Unit, ALU) und ein Steuerwerk. Die durch die Harvard-Architektur vorgesehene physische Trennung von Befehls- und Datenspeicher erfordert zusätzlich eine Speicheradressverwaltung. Ein Prozessor verarbeitet einen Maschinencode *sequentiell*. Der Code wird dabei entweder direkt aus einer Assemblersprache umgewandelt oder durch das Kompilieren von Quelltext in einer höheren Programmiersprache wie Fortran oder C/C++ erzeugt. Zur Verarbeitung von Zahlenwerten wird bis auf ein paar Ausnahmen die Gleitkommaarithmetik (*engl.* Floating-Point Arithmetic) verwendet. Die wesentlichen Prozessortypen zur digitalen Signalverarbeitung auf SDRs sind

- General Purpose Processors (GPP),
- Digital Signal Processors (DSP),
- Advanced RISC Machines (ARM) und
- Graphics Processing Units (GPU).

Die geschickte Implementierung von Algorithmen auf Prozessoren ist bereits ein sorgfältig diskutiertes Thema der Informatik [Eri08]. Es existieren zahlreiche Hilfsmittel und Bibliotheken, die zur Optimierung von Software verwendet werden können. Nichtsdestotrotz stellen die aktuellen, rechenintensiven Entwicklungen in der Nachrichtentechnik, wie z.B. Low Density Parity Check Codes oder Turbo-Verfahren, hohe Anforderung an die Rechenleistung eines Systems. Zur Gewährleistung der Echtzeitfähig sind daher individuelle Implementierungen erforderlich, die die gesamte Rechenkapazität eines Prozessors ausnutzen. Die dazu nötigen Prozessor-spezifischen *Besonderheiten* werden im Folgenden vorgestellt.

Mehrkernprozessoren Zu einer der innovativsten Entwicklungen in der Prozessortechnik hat das Erreichen der sogenannten *Power Wall* geführt. Diese Grenze stellt die maximal mögliche Leistungsaufnahme $P_{\max}$ einer getakteten Halbleiterschaltung dar. Da die Leistungsaufnahme durch den Zusammenhang $P \sim V^2 f$ angenähert werden kann, wird ersichtlich, dass steigende Taktfrequenzen f nur durch das Reduzieren der Versorgungsspannung V erreicht werden können um ein Überhitzen des Prozessors zu verhindern. Durch den Technologieträger und die Halbleiterschaltungstechnik bedingt, ist jedoch die Versorgungsspannung nach unten beschränkt, was zu einer Stagnation der Taktfrequenzen von Prozessoren führte.

Durch die Entwicklung von Mehrkernprozessoren (*engl.* Multi-Core Processor) und die damit geschaffene Parallelität auf Prozessorebene konnte die Leistungsfähigkeit gegenüber konventionellen Einkernsystemen nachhaltig gesteigert werden. Die Anzahl an Kernen und deren Rechenleistung unterscheiden sich dabei je nach Prozessortyp. Das Beispiel in Tabelle 2.1 verdeutlicht dies anhand eines aktuellen GPPs und einer GPU. Auch wenn die einzelnen Kategorien nicht eins-zu-eins

miteinander verglichen werden können, zeigen die teils starken Unterschiede, dass eine Implementierung für den jeweiligen Prozessortyp optimiert werden muss.

Tab. 2.1: Vergleich zwischen GPP und GPU

Merkmal	**GPP**	**GPU**
Prozessorkerne	4	1152
Max. Taktfrequenz	2,6 GHz	1,033 GHz (Boost)
Pufferspeicher	6 MByte	0,288 MByte
Programmierung	Diverse Programmierschnittstellen	CUDA [NCU14]
GPP: Intel Core i7-4960HQ [Ii714]		
GPU: Nvidia GeForce GTX 760 [NGG14]		

Die Herausforderung einer Mehrkernarchitektur ergibt sich bei der Nutzung der Gesamtrechenleistung des Prozessors. Programme müssen parallelisiert werden, um zeitgleich auf Kernen abzulaufen. Der Programmcode wird dazu mithilfe einer Programmierschnittstelle in unabhängige Ausführungsstränge (*engl.* Threads) aufgeteilt. Das Betriebssystem des Prozessors übernimmt anschließend die Ausführungsverwaltung (*engl.* Scheduling) der Threads und teilt diese den verfügbaren Prozessorkernen zu.

Ein Leistungsgewinn gegenüber einem Einkernsystem resultiert aus dem Grad der Parallelisierbarkeit des Programms und der Effizienz der Umsetzung. Als Beispiel wird die Simulationsdauer zur Berechnung der Kreiszahl π nach der Monte-Carlo Methode [BH02] bestimmt. Die Berechnung findet auf einem Intel Core i7 GPP [Ii714] mit vier Prozessorkernen statt und ist zum Vergleich mit und ohne Berücksichtigung der Mehrkernarchitektur umgesetzt. Listing 2.1 zeigt den Programmcode bei Verwendung der Programmierschnittstelle POSIX Threads [Pth95]. Die Ergebnisse in Abbildung 2.3 zeigen, dass durch die Parallelisierung des Algorithmus eine Reduzierung auf fast ein Viertel der Rechenzeit gegenüber einer *naiven* Implementierung ohne Parallelisierung möglich ist.

Das Beispiel demonstriert anschaulich die Bedeutung der Parallelisierung und ihren Stellenwert bei der echtzeitfähigen Umsetzung kom-

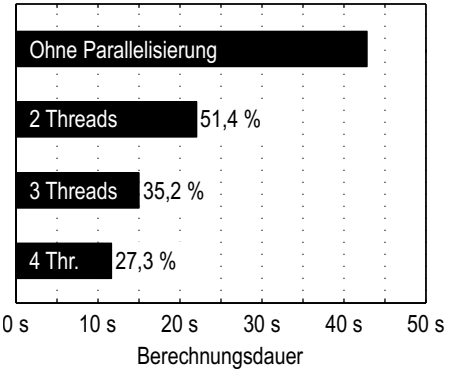


Abb. 2.3: Dauer für die Berechnung der Kreiszahl π nach der Monte-Carlo Methode auf einem Intel Core i7 GPP [Ii714] mit vier Prozessorkernen. Parallelisierung der Berechnung mithilfe POSIX Threads [Pth95]. Insgesamt 10^9 Durchläufe.

plexer Signalverarbeitung. Ebenso zeigt sich, dass weder der Compiler noch das Betriebssystem in der Lage sind *eigenständig* Nebenläufigkeiten im Programmcode zu identifizieren und die Parallelisierung automatisiert durchzuführen. Zur Analyse der Parallelisierbarkeit in Kapitel 6 wird es demnach wichtig sein, Nebenläufigkeiten zu bestimmen bzw. zu erzeugen und ihre Effizienz durch Implementierungen zu verifizieren.

Listing 2.1: Berechnung der Kreiszahl π (POSIX Thread Code-Schnipsel)

```
1  /* Erzeuge alle Threads */
2  for (k=0; k<NUM_THREADS; ++k) {
3
4      /* Uebergebe Daten an Thread k */
5      threads_data[k].seed = time(NULL)+k;
6
7      /* Erzeuge Thread k */
8      pthread_create(&threads[k], &attribute,
9                     thread_fcn,
10                    (void*)&threads_data[k]);
11 }
12
13 /* Warte bis alle Threads fertig sind */
```

```
14 for (k=0; k<NUM_THREADS; ++k) {
15
16     pthread_join(threads[k], NULL);
17 }
```

Als Bewertungsgrößen der Parallelisierbarkeit von Programmen auf Mehrkernprozessoren haben sich *Beschleunigung* und *Effizienz* etabliert. Die Ausführungszeit T_K eines Programms auf einem Prozessor mit K Kernen wird dazu in Relation zur Zeit auf einem Einkernsystem ($K = 1$) gesetzt. Für die Beschleunigung B_K und die Effizienz E_K gilt demnach

$$B_K = \frac{T_1}{T_K} , \tag{2.1}$$

$$E_K = \frac{B_K}{K} . \tag{2.2}$$

Die Ausführungszeiten werden empirisch ermittelt, wobei die kleinste gemessene Zeit aller Simulationsdurchläufe als Referenzwert dient. Dies ermöglicht eine näherungsweise unabhängige Messung von den laufenden Prozessen auf dem Prozessor. Bei der Berechnung der Kreiszahl π (s. Abb. 2.3) beträgt die Beschleunigung bei Nutzung aller Kerne $B_4 = 3{,}66$ und die Effizienz $E_4 = 0{,}915$ mit $T_1 = 42{,}56\,\mathrm{s}$ und $T_4 = 11{,}63\,\mathrm{s}$.

Single Instruction Multiple Data (SIMD) Die als SIMD abgekürzte Rechnerarchitektur ermöglicht eine Parallelisierung auf Datenebene. Da bei einer solchen Architektur die ALU über mehrere unabhängige Recheneinheiten verfügt, können Daten parallel gelesen und während einer Instruktion parallel verarbeitet werden. Darüber hinaus wird durch optimierte Spezialfunktionen (*engl.* Intrinsics) die Berechnung einfacher mathematischer Operationen (+, −, /, exp(·), usw.) ermöglicht. Die aus dem Bereich der Vektor- und Array-Prozessoren stammende Technik, oft auch als Vektorisierung bezeichnet, dient maßgeblich im Multimediabereich zur Verarbeitung großer Datenmengen und ist heutzutage auf verschiedenster, teils proprietärer Art in den genannten Prozessortypen integriert. Bekannte SIMD-Architekturen sind Streaming SIMD Extensions (SSE) von Intel, NEON von ARM oder 3DNow! von AMD.

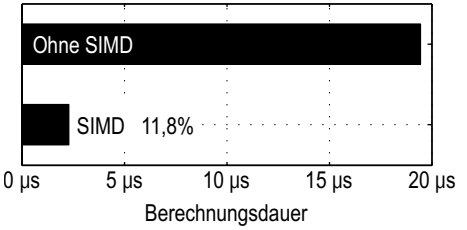


Abb. 2.4: Dauer für die elementweise Berechnung des Betrags für einen komplexwertigen Vektor der Länge 1024 auf einem Intel Core i7 GPP [Ii714] mit AVX-Technologie. Parallelisierung der Berechnung durch VOLK.

Als Beispiel wird die Berechnungsdauer des Betrags für einen komplexwertigen Vektor bestimmt. Die ALU des Intel Core i7 GPPs [Ii714] besitzt eine Breite von 256-bit und unterstützt die SIMD-Technologie Advanced Vector Extensions (AVX), mit der acht 32-bit Zahlen (Datentyp `Float`) parallel verarbeitet werden können. In Listing 2.2 ist der Programmcode gezeigt, der zur Berechnung des Betrags mithilfe von VOLK (Vector-Optimized Library of Kernels) nötig ist. Für jedes Element des komplexwertigen Vektors `c_src` der Länge `N` wird der Betrag berechnet und in `magnitude` gespeichert. Zur Gewährleistung einer hohen Effizienz ist es wichtig, dass die verarbeiteten Vektoren einheitlich, entsprechend der SIMD-Architektur des Prozessors, ausgerichtet (*engl.* Aligned) sind [RMO13], da ansonsten ein zusätzlicher Aufwand bei der Befüllung der SIMD-Register entsteht. Die Ergebnisse in Abbildung 2.4 zeigen, dass durch die SIMD-Architektur die Berechnungsdauer auf ungefähr ein achtel gegenüber einer *naiven* Implementierung gesenkt werden kann.

Listing 2.2: Berechnung des Betrags (VOLK Code-Schnipsel)

```
1 volk_32fc_magnitude_32f_a(magnitude,c_src,N);
```

Die deutlichen Ergebnisse der Datenparallelisierung zeigen, dass auch eine SIMD-Architektur des Prozessors bei der Umsetzung der Turbo-Entzerrung berücksichtigt werden muss. Gegenüber der Parallelisierung für Prozessorkerne gilt es dabei simple, vektorbasierte und datenparallele Operationen innerhalb des Turbo-Entzerrers zu identifizieren und durch geeignete SIMD-Befehle zu ersetzen. Da die Be-

schleunigung, die durch die SIMD-Architektur erreicht werden kann, stark von der Datenwortbreite der verarbeiteten Zahlen abhängt, sei an dieser Stelle auf den Bezug zu der Analyse der Festkommaaspekte in Kapitel 5 hingewiesen. Desto weniger Bits für die Darstellung von Zahlen verwendet werden können, desto größere Gewinne sind bei der Parallelisierung auf Datenebene möglich. Für eine echtzeitfähige Implementierung zeitkritischer Anwendungen gilt es somit, SIMD in Kombination mit minimalen Datenwortbreiten umzusetzen.

2.1.2 Logik

Eine Logik besteht aus einer Vielzahl elementarer Logikgatter, die auf einer Chip-Fläche integriert sind. Die einzelnen Gatter sind aus konfigurierbaren Wahrheitstabellen (*engl.* Lookup-Table, LUT) und getakteten Registern (*engl.* Flipflops) aufgebaut. Durch das Verknüpfen von Logikgattern und das individuelle Programmieren der einzelnen Tabellen wird einem sogenannten Field Programmable Gate Array (FPGA) eine Signalverarbeitung aufgeprägt. Die dafür in einer Netzliste gespeicherten Informationen werden mithilfe eines Synthesewerkzeugs aus einem Quellcode erzeugt. Der Quellcode enthält die Funktion der Schaltung, deren Strukturen und Prozesse, und liegt in einer Hardware-Beschreibungssprache (z.B. Very High Speed Integrated Circuit Hardware Description Language, VHDL) vor. Neuste Generationen von FPGAs verfügen darüber hinaus über dedizierte Logikeinheiten, wie z.B. Multiplizierer oder Microcontroller, zur Implementierung komplexer digitaler Signalverarbeitung. Tabelle 2.2 zeigt exemplarisch relevante Merkmale zweier FPGAs der größten Chip-Hersteller Xilinx und Altera.

Tab. 2.2: Merkmale zweier FPGAs im Vergleich: Xilinx Kintex-7 (XC7K480T) [XK714] und Altera Cyclone V GX (5CGXFC9E5F35C7N) [ACV14]

Merkmal	**Kintex-7**	**Cyclone V**
Anzahl Logikgatter	477760	301000
Pufferspeicher	6,8 MByte	12,2 MByte
Dedizierte Logikeinheiten	1920	342

Ein FPGA ermöglicht aufgrund seiner Architektur eine *parallele* Verarbeitung von Daten. Ähnlich der Mehrkernarchitektur von Prozessoren in Abschnitt 2.1.1 bietet dies erneut die Möglichkeit zur Parallelisierung und effizienten Nutzung der Rechenkapazität. Die hohe Anzahl an Logikgattern (s. Tab. 2.2) und deren, im Vergleich zu physischen Prozessorkernen, beschränkte Rechenleistung, erfordern jedoch eine Analyse der Parallelisierbarkeit auf einer makroskopischen Ebene. Für eine echtzeitfähige Umsetzung auf einen FPGA gilt es demnach erneut die Turbo-Entzerrung hinsichtlich Nebenläufigkeiten zu analysieren und parallele Strukturen zu identifizieren. Kapitel 6 widmet sich dieser Aufgabenstellung.

Die Darstellung von Zahlen findet im Gegensatz zu Prozessoren vorwiegend mithilfe einer Festkommaarithmetik (*engl.* Fixed-Point Arithmetic) statt. Im Folgenden werden die Darstellung und die daraus resultierenden Einschränkungen näher erläutert.

Festkommaarithmetik Die Signalverarbeitung innerhalb einer Logik findet mithilfe einer Festkommaarithmetik statt. Diese wird bereits bei der Maschinen-nahen Beschreibung auf Registertransferebene berücksichtigt [MB07]. Ein Datenwort mit endlicher Anzahl an Bits N repräsentiert einen Zahlenwert mithilfe von 2^N möglichen Amplitudenstufen und schränkt somit den Dynamikbereich der Darstellung ein. Der durch diese Einschränkung entstehende Fehler wirkt sich auf das Verhalten der implementierten Signalverarbeitung aus und muss bei der Umsetzung beachtet werden. Da eine Erhöhung der Stufen zugleich die Anzahl benötigter Gatter steigert, gilt es einen Kompromiss zwischen Dynamik und Flächenverbrauch zu finden. Die Analyse des Fehlers und das Ableiten einer effizienten Festkommadarstellung für die Turbo-Entzerrung sind Gegenstand der Beschreibungen in Kapitel 5.

Die Position des (virtuellen) Kommas innerhalb des Datenworts legt den Wert jedes Bits und somit den darstellbaren Zahlenbereich fest. Von den N Bits des Datenwortes entfallen n Bits auf die Repräsentation der Nachkommastellen. Eine Festkommazahl kann sowohl vorzeichenlos (*engl.* unsigned) als auch vorzeichenbehaftet sein. Die sich daraus ergebene und verbreitete Notation zur Festkommadarstellung

lautet `uFix_N_n` bzw. `Fix_N_n`. Berechnungen in dieser Darstellung werden auf Basis des Zweierkomplements durchgeführt [Flo63]. Tabelle 2.3 gibt den möglichen Wertebereich und die erzielbare Auflösung der Notation an. Für die Zahl 9,76 folgt beispielsweise der Wert 9,75 in der `Fix_7_2` Notation.

Tab. 2.3: Notation, Wertebereich und Auflösung der Festkommadarstellung

Typ	Notation	Wert max.	Wert min.	Auflösung
vorzeichenlos	`uFix_N_n`	$2^{N-n} - 2^{-n}$	0	2^{-n}
vorzeichenbehaftet	`Fix_N_n`	$2^{N-n-1} - 2^{-n}$	-2^{N-n-1}	2^{-n}

Abbildung 2.5 zeigt beispielhaft den Einfluss einer Festkommadarstellung auf die Zahlenwerte einer Sinusschwingung. Während bei der vorzeichenbehafteten Darstellung mit `Fix_4_2` lediglich der Verlauf stark quantisiert wird, werden bei der vorzeichenlosen Darstellung die negativen Werte der Sinusschwingung zu null. Darüber hinaus wurde der Wertebereich beider Darstellungen ($[-2; 1{,}75]$[1] bzw. $[0; 7{,}5]$) nicht optimal an den der Sinusschwingung ($[-1; 1]$) angepasst. All diese Darstellungsfehler wirken sich auf die gesamte nachfolgende Signalverarbeitung aus und führen zur Beeinträchtigung der Leistungsfähigkeit. Aus diesem Grund ist die Analyse von Festkommaaspekten ein wesentlicher Schritt zur Umsetzung der Turbo-Entzerrung in Festkommaarithmetik.

2.2 Zusammenfassung

Die größten Herausforderungen bei der Implementierung digitaler Signalverarbeitung auf SDRs liegen heutzutage in der effizienten Nutzung der parallelen Rechenstrukturen und in der Berücksichtigung der auftretenden Fehler durch eine Festkommaarithmetik. Wenngleich die Rechenleistung eines Logikgatters viel geringer als die eines Prozessorkerns ist, so gilt es jedoch eine Analyse der Parallelisierbarkeit von

[1] Bsp. `Fix_4_2` (s. Tab. 2.3) → Maximalwert: $2^{4-2-1} - 2^{-2} = 1{,}75$; Minimalwert: $-2^{4-2-1} = -2$

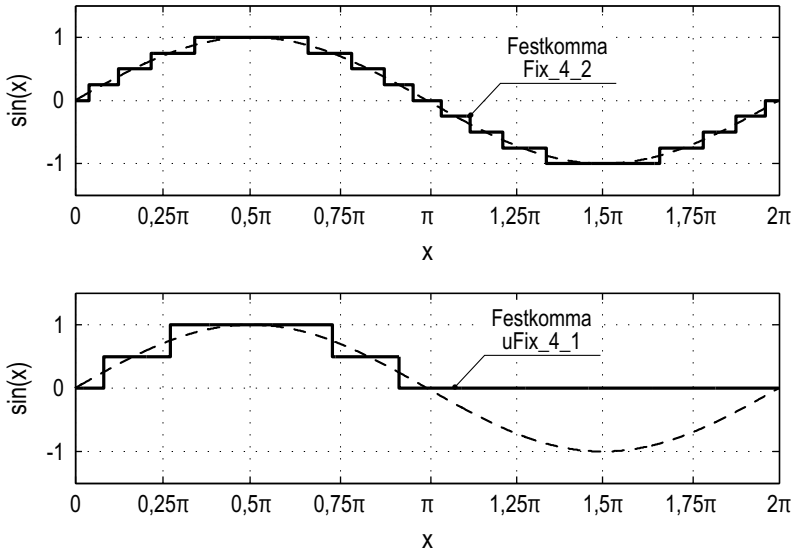


Abb. 2.5: Einfluss der Festkommadarstellung auf die Zahlenwerte einer Sinusschwingung $\sin(x)$. Zur Zuordnung wurde symmetrisches Runden verwendet [BSM01].

Algorithmen für Prozessoren und Logiken durchzuführen. Die Granularität der Untersuchungen muss sich dabei nach der Anzahl und der Rechenkapazität der parallelen Einheiten richten. Obwohl die Festkommadarstellung hauptsächlich bei der Beschreibung von Logiken eine signifikante Rolle spielt, ist eine Analyse der Quantisierungseffekte und das Ableiten effizienter Dimensionierungen auch im Hinblick auf die SIMD-Technologie von großer Bedeutung. Das Reduzieren der Wortbreite spart Logikgatter sowie physikalischen Speicher und ermöglicht darüber hinaus höhere Gewinne bei der Parallelisierung auf Datenebene. Während Kapitel 5 sich der Analyse der Festkommaaspekten widmet, sind in Kapitel 6 die Untersuchungen der Parallelisierbarkeit der Turbo-Entzerrung zusammengefasst.

KAPITEL 3

Digitale Funkübertragung

Die Aufgabe eines digitalen Funksystems besteht in der zuverlässigen Übertragung von Bits über einen Funkkanal durch Modulation einer Trägerschwingung. Die für dieses Ziel nötige Signalverarbeitung in einem Transceiver wird neben den Parametern der Systemkonfiguration, z.B. der Datenrate, vorwiegend durch den Funkkanal bestimmt. In diesem Kapitel wird der Aufbau des betrachteten digitalen Funksystems erläutert und das Systemmodell eingeführt. Das Modell umfasst eine mathematische Beschreibung des zugrunde gelegten Kanals, der senderseitigen Aufbereitung der Information sowie der optimalen Rekonstruktion im Empfänger. Die Ausführungen beschränken sich auf die Vorgänge in der physikalischen Schicht.

Nach der Motivation und Einführung des Kanalmodells in Abschnitt 3.1 wird in Abschnitt 3.2 ein generischer digitaler Sender für Einträgerverfahren vorgestellt. Die Beschreibung des Empfängers in Abschnitt 3.3 bildet den Abschluss des Kapitels und legt den Grundstein zur Einführung der Turbo-Entzerrung in Kapitel 4.

Bemerkungen zur mathematischen Notation

Für die folgenden Ausführungen wird eine, an die gängige Literatur angelehnte, kompakte, mathematische Notation verwendet. Kontinuierliche Zeitabhängigkeiten werden dabei durch Klammern, diskrete Abtastzeitpunkte oder auch Elemente einer Menge, Matrix oder eines Vektors durch einen tiefgestellten Index hervorgehoben. *Beispiele:* $x(t)$ bzw. x_k.

Matrizen und Vektorgen werden durch fett geschriebene Buchstaben angegeben. Ein hochgestelltes H bezeichnet die hermitesche, ein hochgestelltes T die transponierte Matrix. Gleiches gilt für Vektoren. *Beispiele:* $\mathbf{A}$, $\mathbf{A}^H$, $\mathbf{A}^T$, bzw. $\mathbf{a}$, $\mathbf{a}^H$, $\mathbf{a}^T$.

Als besondere Matrizen sind die Einheitsmatrix $\mathbf{I}$, der Nullvektor $\mathbf{0}$, sowie die Fourier-Matrix $\mathbf{F}$ mit der Konstruktionsvorschrift

$$\mathbf{F}_{k,l} = e^{-j\frac{2\pi}{N}kl}, \quad 0 \leq k,l < N, \quad k,l \in \mathbb{N} \cup \{0\}, \tag{3.1}$$

zu nennen. Der Imaginärteil einer komplexen Zahl wird durch $j = \sqrt{-1}$ gekennzeichnet. Für die inverse Fourier-Matrix $\mathbf{B}$ gilt:

$$\mathbf{B} = \frac{1}{N}\mathbf{F}^H \ . \tag{3.2}$$

Abkürzend wird als Notation für Vektoren zusätzlich die Schreibweise

$$(a_k)_{k=0}^{N-1} = \mathbf{a} = (a_0, a_1, \ldots, a_{N-1})^T \ . \tag{3.3}$$

verwendet. Der Faltungsoperator ist durch $*$ gekennzeichnet. Die Notation $\lceil \cdot \rceil$ stellt den Ceiling-Operator dar, der das Argument zur nächst höheren Ganzzahl aufrundet.

3.1 Funkkanal

Das für die Mobilfunkübertragung relevante elektromagnetische Spektrum erstreckt sich von 30 MHz bis 10 GHz. Die physikalischen Effekte,

die sich auf die Ausbreitung der Funkwellen in diesem Bereich auswirken, sind Reflexion, Streuung, Beugung und Brechung. Reflexionen lassen sich an Übergängen zwischen Medien mit unterschiedlichen Wellenwiderständen, z.B. an Häuserwänden, beobachten. Während ein Teil der Welle absorbiert wird, breitet sich der reflektierte Teil phasen- und richtungsverschoben aus. Wird die Energie der Funkwelle durch eine Gruppe von Objekten, z.B. durch Blätter in einen Raumwinkelbereich reflektiert, spricht man von Streuung. Die auf dem Huygens'schen Prinzip [Ger10] beruhende Ausleuchtung von Objektschatten wird als Beugung bezeichnet. Die Wellenbrechung ist auf die Änderung der Ausbreitungsgeschwindigkeit zurückzuführen. Ähnlich zur Reflexion führt dies zur Ausprägung neuer Wellenkomponenten. Weiterführende Literatur zum Thema Mobilfunkkanäle bietet [PM99].

Wird eine Funkübertragung über einen solchen Kanal durchgeführt, lassen sich im Empfänger verschiedene Einflüsse auf das Signal erkennen. Die physikalischen Effekte führen dabei zur Ausprägung mehrerer Signalwege (Mehrwegeausbreitung) zwischen Sender und Empfänger, die sowohl unterschiedlich gedämpft sind als auch in ihrer Phase und Laufzeit variieren. Die so entstehende Interferenz am Empfänger ist abhängig von der Frequenz und kann destruktiv auf das Signal wirken (*engl.* Fading). Die Intensität sowie die zeitliche Varianz ist dabei maßgeblich von der Topografie, der Morphografie, der Bebauung und nicht zuletzt von der Systemkonfiguration abhängig. Sind zudem Sender oder Empfänger mobil, ergibt sich aufgrund des Dopplereffekts eine zusätzliche Zeitabhängigkeit der Effekte. Zusammenfassend wird im Folgenden die Bezeichnung *frequenzselektiver Fading-Kanal* zur Klassifizierung der erläuterten Ausbreitungseffekte verwendet.

3.1.1 Mathematische Beschreibung

Zur Herleitung der mathematischen Kanalbeschreibung wird die Signaldarstellung im komplexen Basisband gewählt. Der frequenzselektive Fading-Kanal wird als *lineares* Übertragungssystem modelliert und ist vollständig durch die zeitvariante Impulsantwort $h(t, \tau)$ definiert.

Das Empfangssignal $r(t)$ resultiert nun aus der Faltung des Sendesignals $s(t)$ mit der Kanalimpulsantwort und der Überlagerung durch das Empfängerrauschen $n(t)$

$$r(t) = \int_{-\infty}^{\infty} h(t,\tau)s(t-\tau)\,\mathrm{d}\tau + n(t)\ . \tag{3.4}$$

Das Rauschen wird als Realisierung eines Zufallsprozesses mit komplexer, zirkulärsymmetrischer Gauß-Verteilung modelliert. Es ist mittelwertfrei, unkorreliert und besitzt die Varianz σ_0^2

$$n(t) \sim \mathcal{CN}\left(0,\sigma_0^2\right), \quad \mathrm{E}\left\{n\left(t'\right) * n\left(t''\right)\right\} = \sigma_0^2\delta(t'-t'')\ . \tag{3.5}$$

Die Digitalisierung des Empfangssignals führt zu einer zeitlich begrenzten Auflösung. Infolgedessen können Signalpfade mit Laufzeitdifferenzen im Bereich der Abtastzeit T_a nicht mehr voneinander getrennt werden. Für die Beschreibung des Kanals folgt somit eine Diskretisierung der Impulsantwort gemäß

$$h(t,\tau) = \sum_{p=-\infty}^{\infty} h\left(t,pT_a\right)\mathrm{sinc}\left(\frac{\pi}{T_a}\left(\tau - pT_a\right)\right)\ . \tag{3.6}$$

Die diskreten Kanalkoeffizienten $h_p(t) := h\left(t,pT_a\right)$ stellen nun den Einfluss einer Gruppe von Signalpfaden dar, die ungefähr mit der Laufzeit $p \cdot T_a$ am Empfänger eintreffen. Wird zudem angenommen, dass die Impulsantwort nach endlicher Dauer $P \cdot T_a$ unterhalb einer Dämpfungsgrenze abgeklungen und kausal ist, folgt die diskrete Kanalimpulsantwort in Vektordarstellung zu

$$\mathbf{h}(t) = \left(h_0(t), h_1(t), \ldots, h_p(t), \ldots, h_{P-1}(t)\right)^T\ , \tag{3.7}$$

mit $h(t,\tau) = 0$ für $\tau < 0$. Diese Form der Kanalmodellierung wird als Tapped-Delay-Line (TDL) [PM99] bezeichnet und ist in Abbildung 3.1 dargestellt.

3.2 Sender

Der in dieser Arbeit betrachtete digitale Sender besitzt einen generischen Aufbau und ist in vielen modernen Funkstandards, wie z.B.

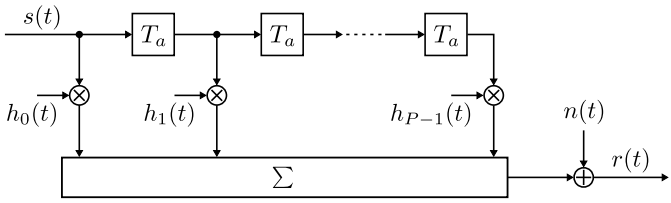


Abb. 3.1: Tapped-Delay-Line Modell des Funkkanals

LTE, wiederzufinden. Die Kernkomponenten sind ein Encoder zum Fehlerschutz, ein Interleaver zum Schutz gegen Bündelfehler sowie eine Modulation zum Aufprägen der Bits auf eine Trägerschwingung. Diese Form der Informationsaufbereitung wird als Bit-Interleaved Coded Modulation (BICM) bezeichnet [CTB98]. Als ein Resultat der Paketorientierten Verarbeitung in der Vermittlungs- und Transportschicht heutiger Kommunikationssysteme wird eine Block-basierte Übertragung von Bits durchgeführt. Das Blockschaltbild des BICM-Senders ist in Abbildung 3.2 dargestellt. Weiterführende Literatur zu diesem Thema findet sich in [Kam04], [Gol05] oder [PS07].

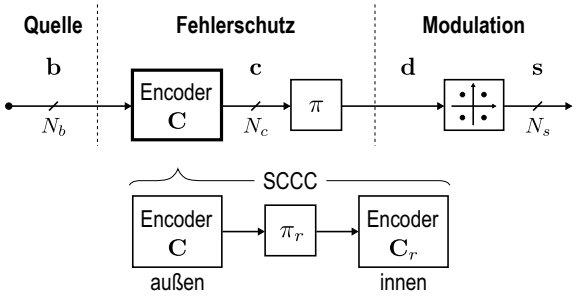


Abb. 3.2: Blockschaltbild des BICM-Senders

3.2.1 Quelle

Die zu übertragenden Infobits b_k werden durch eine Quellencodierung vorverarbeitet und sind unkorreliert. Ebenso ist die Wahrscheinlichkeit

für das Auftreten eines 0 oder 1 Bits gleich groß. Die Bits werden zur Weiterverarbeitung zu Sequenzen der Länge N_b zusammengefasst. Im Folgenden wird die Verarbeitung einer einzelnen Sequenz

$$\mathbf{b} = (b_0, \ldots, b_{N_b-1})^T \tag{3.8}$$

betrachtet, womit eine explizite Kennzeichnung der Zeitabhängigkeit entfällt. Die Energie pro Bit ist auf $E_b = 1$ normiert. Das primäre Ziel des Empfängers ist die zuverlässige Rekonstruktion dieser Infobits.

3.2.2 Fehlerschutz

Zum Schutz gegen Bitfehler werden die Infobits durch einen Encoder (Kanalcodierung) und einen Interleaver vorverarbeitet. Im Encoder kommen dabei die in der Mobilkommunikation gebräuchlichen und leistungsfähigen Faltungscodes [Bos13] zum Einsatz. Diese werden durch Generatorpolynome beschrieben und mihilfe eines Schieberegisters der Länge ν implementiert. Während einer Encodierung werden n_b Infobits auf n_c Codebits gemäß des vorgegebenen Codes $\mathbf{C}$ abgebildet. Die sich daraus ergebene Coderate $r = \frac{n_b}{n_c}$ gilt als Maß für die induzierte Redundanz und erfüllt $0 < r \leq 1$. Vorgreifend auf die Signalverarbeitung im Empfänger wird der Encoder durch Manipulation der Infobit-Sequenz in einen definierten Endzustand terminiert [Bos13]. Die Notwendigkeit dieser Maßnahme wird in Abschnitt 4.3.3 erläutert. Die durch den Encoder erzeugte Codebit-Sequenz ist im Folgenden durch

$$\mathbf{c} = (c_0, \ldots, c_{N_c-1})^T \tag{3.9}$$

gegeben. Die für diese Arbeit relevanten Familien von Faltungscodes werden nun kurz vorgestellt.

Single Convolutional Code (SCC) Diese konventionelle Form der Kanalcodierung besteht aus einem einzigen Faltungscode. Es werden im Weiteren Rate-Compatible Punctured Convolutional (RCPC) Codes [Bos13] verwendet, bei der ein *Mutter*-Code mit der Rate $\frac{1}{n_c}$ arbeitet. Codes mit höherer Rate werden durch Punktierung [Bos13]

erzeugt. Der Vorteil dieser RCPC-Codes liegt in der einfachen Variation der Coderate ohne Änderung des Encoders und Decoders. In der Veröffentlichung [Hag88] von Hagenauer aus dem Jahre 1988 wird gezeigt, dass RCPC-Codes nahezu die gleiche Korrekturfähigkeit wie Faltungscodes vergleichbarer Rate aufweisen.

Zur Beschreibung von SCCs wird ein Konstrukt aus mehreren Vektoren verwendet, die jeweils die Koeffizienten der Generatorpolynome [Bos13] repräsentieren

$$\mathbf{C} = [\mathbf{g}_0, \ldots, \mathbf{g}_{n_c-1}; \mathbf{g}_r] \,. \tag{3.10}$$

Der optionale Vektor $\mathbf{g}_r$ dient zur Konstruktion rekursiver Codes und gibt die rückgekoppelten Zustände des Encoders an. Abbildung 3.3 zeigt den Encoder eines rekursiven Rate-$\frac{1}{2}$ SCCs mit $\mathbf{C} = [15, 17; 13]_8$. Mit der tiefgestellten 8 wird üblicherweise gekennzeichnet, dass die Zahlenwerte im Oktalsystem (Stellenwertsystem mit Basis 8) angeben sind.

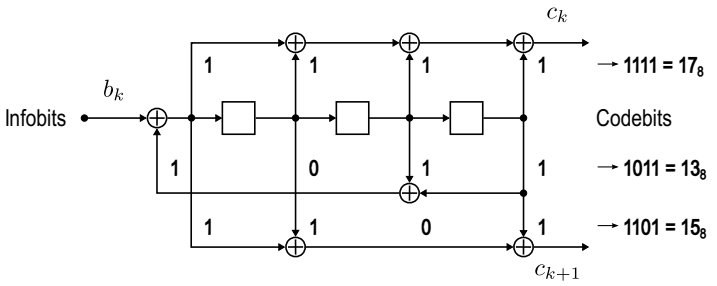


Abb. 3.3: Encoder eines Rate-$\frac{1}{2}$ SCCs mit Code $\mathbf{C} = [15, 17; 13]_8$

Serial Concatenated Convolutional Code (SCCC) Das Konzept zweier seriell verketteter Faltungscodes, die durch einen Interleaver π_r voneinander getrennt sind, wurde erstmals 1998 von Benedetto *et al.* vorgeschlagen [BDMP98]. Für die Turbo-Entzerrung sind dabei besonders die Kombinationen relevant, bei denen der *äußere* Code durch einen SCC und der *innere* durch einen rein rekursiven Rate-1 Code $\mathbf{C}_r$ realisiert werden. Durch den inneren Faltungscode werden

die Dispersionseigenschaften des Interleavings verbessert und damit die Leistungsfähigkeit der Turbo-Entzerrung gegenüber konventionellen Encodern gesteigert [Tü04][Gro11].

Interleaver Im Empfänger entstehen Bündelfehler bei der Entscheidung, wenn das Funksignal für mehrere aufeinanderfolgende Infobits gestört wurde. Diese Störungen entstehen durch Einbrüche der Empfangsleistung aufgrund von Abschattungseffekten des Kanals. Da die Fehler zeitlich korreliert sind, werden hohe Anforderung an die Korrekturfähigkeit und somit an die Komplexität des verwendeten Faltungscodes gestellt. Zur Aufwandsreduzierung wird deshalb im Sender eine Permutation der Bits innerhalb der Codebit-Sequenz vorgenommen. Das Permutationsschema $\pi: \{0, \ldots, N_c - 1\} \rightarrow \{0, \ldots, N_c - 1\}$ ist im Empfänger bekannt und liefert

$$d_k = c_{\pi(k)} \ , \quad k = 0, \ldots, N_c - 1 \ . \tag{3.11}$$

Die neu geordnete Sequenz wird im Folgenden als Datenbit-Sequenz

$$\mathbf{d} = (d_0, \ldots, d_{N_c-1})^T \tag{3.12}$$

bezeichnet. Eine erneute Permutation vor dem Entscheider mit dem inversen Schema $\pi^{-1}: \{0, \ldots, N_c - 1\} \rightarrow \{0, \ldots, N_c - 1\}$ führt zur Verteilung der Bündelfehler und zur Erzeugung (näherungsweise) statistisch unabhängiger Einzelfehler.

3.2.3 Modulation

Das Aufprägen der Information auf eine Trägerschwingung wird im Basisband durch das Abbilden (*engl.* Mapping) der Datenbits auf Symbole eines M-stufigen Modulationsalphabets $\mathcal{S}$ mit der Codierungsvorschrift $\mathcal{C}$ (z.B. Gray-Codierung [Kam04]) realisiert. Die in modernen Funkstandards eingesetzten und in dieser Arbeit verwendeten linearen Modulationsverfahren, z.B. Phasenumtastung, sind gedächtnislos und vollständig durch die Konstellation ihrer komplexwertigen Symbole definiert. Zur Abbildung werden jeweils $\mathrm{ld}(M)$ Datenbits zu einer

Bitgruppe zusammengefasst und das entsprechende Symbol mittels einer Lookup-Table oder Konstruktionsvorschrift bestimmt. Die Länge der Symbol-Sequenz

$$\mathbf{s} = \left(s_0, \ldots, s_{N_s-1}\right)^T \quad \in \mathbb{C}^{N_s} \tag{3.13}$$

ist durch $N_s = N_c/\mathrm{ld}(M)$ gegeben. Die mittlere Leistung der Sendesymbole wird ohne Beschränkung der Allgemeinheit auf $\mathrm{E}\{|s_k|^2\} = 1$ festgelegt. Abbildung 3.4 zeigt das Signalraumdiagramm für QPSK und 16-QAM mit Gray-Codierung.

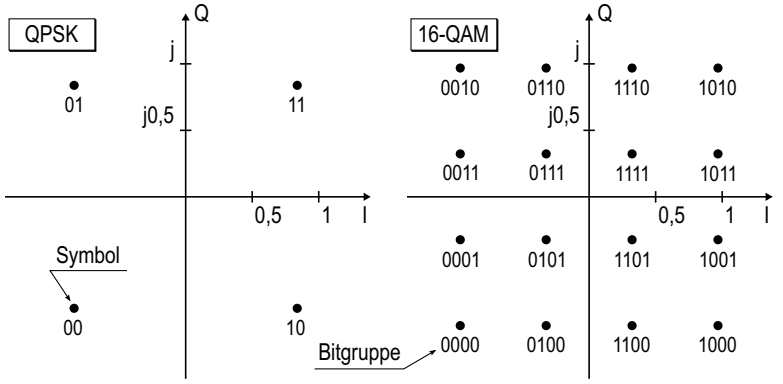


Abb. 3.4: Signalraumdiagramm für QPSK und 16-QAM mit Gray-Codierung

Um eine Block-unabhängige Signalverarbeitung im Empfänger zu ermöglichen, werden benachbarte Blöcke im Sender durch Schutzintervalle getrennt. Die minimale Länge des Intervalls entspricht dabei der Dauer[1] des längsten Mehrwegepfads $T_s \cdot (P - 1)$. Es existieren verschiedene Ansätze den Inhalt des Schutzintervalls zu konzipieren. Im Hinblick auf die Frequenzbereichsentzerrung wird das zyklische Präfix (*engl.* Cyclic-Prefix, CP) gewählt [Kam04]. Das Schutzintervall ist dabei mit den letzten $P-1$ Symbolen jeder Symbol-Sequenz belegt.

[1] Es wird angenommen, dass zur Funkübertragung die volle Systembandbreite verwendet wird. Die Symboldauer T_s entspricht demnach der Abtastzeit T_a.

3.3 Empfänger

Die durch den frequenzselektiven Fading-Kanal bedingten Laufzeiten und Phasenverschiebungen der Signalpfade führen zu Eigeninterferenz (Intersymbolinterferenz, *engl.* Intersymbol Interference (ISI)) des Funksignals. Zur Rekonstruktion der gesendeten Bits muss der Einfluss des Kanals im Empfänger ermittelt und rückgängig gemacht werden. Für den mit Kanalentzerrung (*kurz* Entzerrung) bezeichneten Vorgang existieren verschiedene Verfahren. Erste Ansätze gehen auf das Jahr 1860, das Zeitalter kabelgebundener Telegraphen, zurück [Fal11]. Der Fortschritt der Elektronik, die Entwicklung schnellerer Schaltkreise, der Übergang von Analog- zu Digitaltechnik, haben sich dabei ebenso auf die Funktechnik wie auch auf die Forschung im Bereich der Entzerrung ausgewirkt. Nichtlineare, adaptive oder blinde Verfahren zur Entzerrung sowie Methoden zur Reduzierung der Komplexität sind weitestgehend untersucht und in vielen Funksystemen implementiert. Die in dieser Arbeit diskutierte Turbo-Entzerrung stellt demgegenüber ein Verfahren dar, welches den Fehlerschutz im Signal zur Entzerrung ausnutzt.

Im Folgenden wird die mathematische Beschreibung des Empfangssignals eingeführt. Die Darstellung des Optimalempfängers bildet anschließend die Basis zur Erläuterung der Turbo-Entzerrung in Kapitel 4. Da der Fokus dieser Arbeit auf der Entzerrung des Funksignals liegt, wird die Synchronisation im Empfänger als perfekt vorausgesetzt und auf die einschlägige Literatur über leistungsfähige Ansätze zur Zeit- und Frequenzsynchronisation verwiesen, z.B. [Men97], [Gol05], [TV05] oder [PS07].

3.3.1 Empfangssignal

Für die folgenden Herleitungen wird erneut die Signaldarstellung im komplexen Basisband gewählt. Das Empfangsfilter sei auf das Sendefilter angepasst und die Abtastung des Ausgangs finde synchron zum Sender statt. Die Kettenschaltung aus Sende- und Empfangsfilter erfüllt die *erste* Nyquist-Bedingung. Des Weiteren wird neben der perfekten Synchronisation von Zeit und Frequenz die volle Kenntnis

des Kanals (*engl.* Channel State Information at the Receiver, CSIR) und des Senderaufbaus vorausgesetzt. Jede Komponente im Empfänger kennt somit $\mathbf{H}$, σ_0^2, $\mathbf{C}$, π und eventuell $\mathbf{C}_r$ und π_r. Störeinflüsse, die durch Ungenauigkeiten elektronischer Bauteile hervorgerufen werden, sind nicht Bestandteil der Analyse.

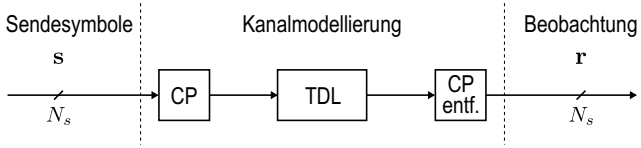


Abb. 3.5: Übertragungsmodell

Unter diesen Voraussetzungen sowie den Ausführungen in Abschnitt 3.1 und 3.2 kann der frequenzselektive Fading-Kanal durch eine diskrete Tapped-Delay-Line modelliert werden (s. Abb. 3.5). Die Dauer der Kanalbeobachtung wird ohne Beschränkung der Allgemeinheit entsprechend der Dauer eines Sendeblocks gewählt. Darüber hinaus wird angenommen, dass die Kohärenzzeit des Funkkanals größer als die Dauer eines Sendeblocks ist. Für den in der Literatur mit *Block-Fading* charakterisierten Kanal werden demnach die Kanalkoeffizienten während einer Beobachtung als zeitinvariant $h_p := h_p(t)$ betrachtet.

Nach dem Entfernen des zyklischen Präfixes sind die Empfangssymbole $\mathbf{r}$ durch

$$\mathbf{r} = \mathbf{H}\mathbf{s} + \mathbf{n} \quad \in \mathbb{C}^{N_s} \tag{3.14}$$

gegeben. Das in Abschnitt 3.1.1 eingeführte Gauß'sche Rauschen ist zirkulärsymmetrisch und wird durch den Vektor

$$\mathbf{n} = \left(n_0, n_1, \ldots, n_k, \ldots, n_{N_s-1}\right)^T, \quad \in \mathbb{C}^{N_s}, \tag{3.15}$$

mit $\mathrm{E}\{\mathbf{n}\} = \mathbf{0}$ und $\mathrm{E}\{\mathbf{n}\mathbf{n}^H\} = \sigma_0^2\mathbf{I}$ beschrieben. Im Weiteren wird das Verhältnis von E_b/σ_0^2 als Signal-to-Noise Ratio (SNR) pro Bit bezeichnet. Der Aufbau der zyklischen Kanalfaltungsmatrix $\mathbf{H} \in \mathbb{C}^{N_s \times N_s}$ lau-

tet

$$\mathbf{H} = \begin{pmatrix} h_0 & 0 & \cdots & 0 & h_{P-1} & \cdots & h_1 \\ h_1 & h_0 & 0 & \cdots & 0 & h_{P-1} & \cdots \\ \cdots & h_1 & h_0 & 0 & \cdots & 0 & h_{P-1} \\ h_{P-1} & \cdots & h_1 & h_0 & 0 & \cdots & 0 \\ 0 & h_{P-1} & \cdots & h_1 & h_0 & 0 & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & h_{P-1} & \cdots & h_1 & h_0 \end{pmatrix} . \tag{3.16}$$

3.3.2 Optimalempfänger

Der hinsichtlich minimaler Bitfehlerwahrscheinlichkeit optimale Empfänger für das hier betrachtete Systemmodell basiert auf der Maximum *a posteriori* (MAP) Entscheidung [Kro04]. Dieser Optimalempfänger entscheidet zugunsten desjenigen Infobits $\hat{b}_k$, das bei gegebener Beobachtung $\mathbf{r}$ mit größter Wahrscheinlichkeit gesendet wurde

$$\hat{b}_k = \underset{b \in \{0,1\}}{\arg\max}\, P\left(b_k = b \,|\, \mathbf{r}\right), \quad k = 0, \ldots, N_b - 1 . \tag{3.17}$$

Da für diese Entscheidung alle 2^{N_b} möglichen Infobit-Sequenzen ausgewertet werden müssen, wächst die Komplexität dieses Ansatzes mit $\mathcal{O}\left(2^{N_b}\right)$. Nach heutigem Stand der SDR-Technik ist dies für hohe Sequenzenlängen nicht in Echtzeit implementierbar.

Ausgehend von diesem Ansatz lassen sich jedoch die nötigen Schritte zur Turbo-Entzerrung herleiten und motivieren. Zunächst wird dazu die MAP-Entscheidungsregel mithilfe eines Log-Likelihood Ratios (LLR) dargestellt. Angewendet auf die *a posteriori* Wahrscheinlichkeit in (3.17) gilt für ein LLR

$$L(b_k \,|\, \mathbf{r}) = \ln\left(\frac{P\left(b_k = 1 \,|\, \mathbf{r}\right)}{P\left(b_k = 0 \,|\, \mathbf{r}\right)}\right) \in \mathbb{R} . \tag{3.18}$$

Während die Amplitude des LLRs die Zuverlässigkeit der Entscheidung (*engl.* Soft-Decision) widerspiegelt, erlaubt das Vorzeichen eine

harte Entscheidung (*engl.* Hard-Decision) gemäß

$$\hat{b}_k = \begin{cases} 1, & L(b_k \,|\, \mathbf{r}) \geq 0 \\ 0, & L(b_k \,|\, \mathbf{r}) < 0 \end{cases} . \tag{3.19}$$

Im nächsten Schritt lässt sich die Wahrscheinlichkeit in (3.17) zerlegen und mithilfe des Satzes von Bayes [JW02] umformen in

$$\begin{aligned} P\left(b_k = b \,|\, \mathbf{r}\right) &= \sum_{\mathbf{b}:\, b_k = b} P(\mathbf{b} \,|\, \mathbf{r}) \\ &= \sum_{\mathbf{b}:\, b_k = b} \frac{p(\mathbf{r} \,|\, \mathbf{b}) P(\mathbf{b})}{p(\mathbf{r})} . \end{aligned} \tag{3.20}$$

Da von der Unabhängigkeit der gesendeten Infobits ausgegangen wird (s. Abs. 3.2.1), faktorisiert $P(\mathbf{b})$ in

$$P(\mathbf{b}) = \prod_{l=0}^{N_b - 1} P(b_l) . \tag{3.21}$$

Gleichung (3.20) und (3.21) ermöglichen es als nächstes (3.18) durch

$$L(b_k \,|\, \mathbf{r}) = \ln \left(\frac{\sum\limits_{\mathbf{b}:\, b_k = 1} p\left(\mathbf{r} \,|\, \mathbf{b}\right) \prod_{l=0}^{N_b - 1} P(b_l)}{\sum\limits_{\mathbf{b}:\, b_k = 0} p\left(\mathbf{r} \,|\, \mathbf{b}\right) \prod_{l=0}^{N_b - 1} P(b_l)} \right) \tag{3.22}$$

auszudrücken, wobei der Term $p(\mathbf{r})$ bereits gekürzt wurde. Im letzten Schritt kann die Wahrscheinlichkeit $P(b_k = 1)$ bzw. $P(b_k = 0)$ aus dem Produkt herausgezogen werden. Der fundamentale Ansatzpunkt für die Turbo-Entzerrung lautet damit schließlich [HLY02][TKS02]

$$\begin{aligned} L(b_k \,|\, \mathbf{r}) &= \underbrace{\ln \left(\frac{\sum\limits_{\mathbf{b}:\, b_k = 1} p\left(\mathbf{r} \,|\, \mathbf{b}\right) \prod_{l=0:\, l \neq k}^{N_b - 1} P(b_l)}{\sum\limits_{\mathbf{b}:\, b_k = 0} p\left(\mathbf{r} \,|\, \mathbf{b}\right) \prod_{l=0:\, l \neq k}^{N_b - 1} P(b_l)} \right)}_{= \lambda(b_k)} + \underbrace{\ln \left(\frac{P(b_k = 1)}{P(b_k = 0)} \right)}_{= \zeta(b_k)} \\ &= \lambda(b_k) + \zeta(b_k) . \end{aligned} \tag{3.23}$$

Dieser Zusammenhang verdeutlicht, dass sich die Zuverlässigkeit der Entscheidung im Empfänger zugunsten eines Infobits b_k aus zwei unabhängigen LLRs zusammensetzt. Das mit $\lambda(\cdot)$ bezeichnete LLR stellt den Beitrag zur Zuverlässigkeit dar, der ausschließlich aufgrund der Infobits b_l, mit $l \neq k$, in der Beobachtung existiert. Hinsichtlich der fehlenden Abhängigkeit zu b_k wird dieser Term als *extrinsische*[1] Information bezeichnet bzw. im Kontext der Turbo-Entzerrung auch als kombinierte Kanal- und extrinsische Information [HLY02]. Demgegenüber entspricht das zweite LLR $\zeta(\cdot)$ der Zuverlässigkeit, die exklusiv von den Wahrscheinlichkeiten von b_k abhängt. Sie wird als *a priori* Information bezeichnet. Ist es nun möglich, dieses Wissen über die Quelle zu erzeugen, kann dadurch die MAP-Entscheidung verbessert werden. Das ist die prinzipielle Idee hinter der Turbo-Entzerrung, welche im folgenden Kapitel 4 detailliert erläutert wird.

Bemerkung: Im Gegensatz zu Turbo-Codes, bei der das *a posteriori* LLR gewöhnlich in drei unabhängige LLRs aufteilt werden kann

$$L(b_k \,|\, \mathbf{r}) = \underbrace{L(b_k \,|\, r_k)}_{\text{intr.}} + \underbrace{L(b_k \,|\, \mathbf{r}_{\setminus k})}_{\text{extr.}} + \underbrace{L(b_k)}_{a\ priori} \,, \tag{3.24}$$

mit $\mathbf{r}_{\setminus k} = (r_0, \ldots, r_{k-1}, r_{k+1}, \ldots, r_{N_s-1})$, ist dieses Vorgehen für das hier betrachtete Systemmodell und für die Turbo-Entzerrung im Allgemeinen nicht möglich. Die durch den frequenzselektiven Fading-Kanal verursachte zeitliche Dispersion der Symbole (s. Gl. (3.14)) verbietet eine explizite Aufteilung des LLRs $L(b_k \,|\, \mathbf{r})$ in die intrinsische und extrinsische Komponente [HLY02].

[1] Bedeutung: *von außen her*

KAPITEL 4

Turbo-Entzerrung

Das auf dem *Turbo*-Prinzip beruhende Verfahren zur Entzerrung von Signalen mit Eigeninterferenz wurde erstmals 1995 von Douillard *et al.* [DJB+95] vorgeschlagen. Der Ansatz beruht auf der Aufteilung des Optimalempfängers aus Abschnitt 3.3.2 in die dedizierten Komponenten *Entzerrer* und *Decoder*. Die Komponenten werden durch MAP-basierte Algorithmen realisiert und über einen Interleaver bzw. Deinterleaver miteinander verknüpft. In einem iterativen Prozess tauschen Entzerrer und Decoder nach der Beobachtung eines Empfangssignals extrinsische Information aus, die der Entscheidungszuverlässigkeit der jeweiligen Komponente bezüglich einer gesendeten Codebit- bzw. Datenbit-Sequenz entspricht. Beide Komponenten müssen in der Lage sein sowohl Soft-Input zu verarbeiten als auch Soft-Output zu liefern, was die Bezeichnung SISO-Komponenten geprägt hat. Die Leistungsfähigkeit der Turbo-Entzerrung liegt darin begründet, dass mit jeder Iteration die Zuverlässigkeit steigt und somit die Anzahl der Bitfehler sinkt. Eine Zusammenfassung über Turbo-basierte Systeme bietet [HLY02].

Seit der ersten Veröffentlichung sind bereits zahlreiche Publikationen entstanden, die sich vorwiegend mit der Entzerrer-Komponente des Verfahrens beschäftigen. Der Grund hierfür liegt in der hohen Rechenkomplexität einer MAP-basierten Entzerrung. Die Anzahl der möglichen Zustände hängt exponentiell von der Modulationsstufigkeit M und der Länge der Kanalimpulsantwort P ab und beträgt M^P. Für

stark dispersive Kanäle und Modulationsverfahren hoher Ordnung ist eine solche Entzerrung nicht in Echtzeit umsetzbar[1]. Der Fokus der Untersuchungen liegt deshalb auf der Konstruktion und Analyse suboptimaler SISO-Entzerrer, die sich als Ersatz im Turbo-Entzerrer eignen.

Aus Sicht der Implementierung und Leistungsfähigkeit haben sich dabei die 2002 von Tüchler, Koetter und Singer untersuchten linearen Filter-basierten Ansätze mit Minimum Mean Square Error (MMSE) Schätzer als bedeutsam erwiesen [TKS02][TSK02]. Diese Arbeit geht wiederum auf die Ergebnisse einer Veröffentlichung von Wang und Poor [WP99] zurück, die 1999 im Kontext von Code Division Multiple Access (CDMA) Systemen entstanden. Das ursprüngliche Ziel, die Unterdrückung der Interferenz durch andere CDMA-Nutzer mittels Soft Interference Cancellation (SC), wurde für die Turbo-Entzerrung von Tüchler *et al.* in die Unterdrückung der Eigeninterferenz umformuliert. Mit Respekt vor den theoretischen Arbeiten, die sich beispielsweise auf Trellis-basierte Entzerrer (z.B. [FBC07], [KWW08] oder [PCFB10]) konzentrieren oder klassische Entzerrer (z.B. [GLL97] oder [LB06]) adaptieren und integrieren, wird in dieser Arbeit das SC-MMSE Verfahren aufgrund seiner hohen Leistungsfähigkeit und geringen Komplexität verwendet.

Eine weitere Methode zur Reduzierung der Rechenkomplexität ist die Frequenzbereichsentzerrung. Basierend auf dem Faltungstheorem der Fourier-Transformation [KK06] und effizienten Implementierungen der Diskreten Fourier-Transformation (DFT) können digitale Filter mit geringer Anzahl an Multiplikationen und Additionen realisiert werden. Die Ressourcenersparnis durch eine Entzerrung im Frequenzbereich (*engl.* Frequency Domain, FD) im Vergleich zur Zeitdomäne ist ab einer gewissen Filterlänge gegeben und abhängig vom verwendeten DFT-Algorithmus[2]. Im Zuge der Entwicklung breitbandiger Mobilfunksysteme, wie z.B. LTE, wurde die ursprüngliche Idee von Walz-

[1] Bsp.: 16-QAM ($M = 4$) und $P = 24 \rightarrow M^P = 4^{24} = 2{,}8 \cdot 10^{14}$ Zustände

[2] Bsp.: Eine Radix-2 FFT der Länge 1024 benötigt 4,5 komplexe Multiplikationen pro Eingangswert (s. Abs. 6.2), d.h. insgesamt sind $4{,}5 + 1 + 4{,}5 = 10$ komplexe Multiplikationen zur Entzerrung nötig. Demnach ist eine Frequenzbereichsentzerrung unter diesen Randbedingungen gegenüber einer Zeitbereichsentzerrung für mehr als 10 komplexe Filterkoeffizienten effizienter.

man und Schwarz [WS73] in den vergangenen Jahren erneut aufgegriffen. Die viel zitierten Veröffentlichungen von Sari *et al.* [SKJ95] und Falconer *et al.* [FABSE02] beschäftigen sich z.B. mit der Frequenzbereichsentzerrung breitbandiger TV-Rundfunksignale und führen einen Vergleich mit dem gängigen Orthogonal Frequency-Division Multiplexing (OFDM) durch.

Die Analyse der Turbo-Entzerrung hinsichtlich Implementierungsaspekten findet in dieser Arbeit für Einantennensysteme (*engl.* Single Input Single Output) statt. Nichtsdestotrotz lassen sich die Ergebnisse auch qualitativ auf Mehrantennensysteme (*engl.* Multiple Input Multiple Output, MIMO) übertragen, da ein MIMO-Turbo-Entzerrer für einen Sender mit N_t Antennen im Empfänger aus N_t parallelen Turbo-Entzerrern aufgebaut ist [ATM03][KM07][Gro12]. Die Unterschiede beider Systeme liegen in der Konstruktion der MMSE-Entzerrermatrix und in der Komplexität des Soft Interference Cancellations.

Im Folgenden wird das Prinzip der klassischen Turbo-Entzerrung für Single Convolutional Codes (SCC) näher erörtert und in Abschnitt 4.1 der grundlegende Ablauf unter Verwendung MAP-basierter Komponenten verdeutlicht. Das in dieser Arbeit verwendete SC-MMSE-FD Verfahren ist Gegenstand der Beschreibung in Abschnitt 4.2. Abschnitt 4.3 widmet sich dem SISO-Decoder im Turbo-Entzerrer und erläutert das Prinzip der Max*-Log-MAP Decodierung. Abschnitt 4.3.5 befasst sich mit den nötigen Modifikationen hinsichtlich der Decodierung von Serial Concatenated Convolutional Codes (SCCC). Die Analyse des Turbo-Entzerrers mithilfe von Extrinsic Information Transfer (EXIT) Diagrammen bildet eine effiziente Darstellung des Konvergenzverhaltens und wird in Abschnitt 4.4 beschrieben. Abschließend verdeutlicht Abschnitt 4.5 die Leistungsfähigkeit der Turbo-Entzerrung für verschiedene Kanäle.

4.1 Funktionsweise

Der erste Schritt hin zur Turbo-Entzerrung ist das Aufteilen des in Abschnitt 3.3.2 beschriebenen optimalen Empfängers in zwei isolierte MAP-Komponenten. Die erste SISO-Komponente berechnet aus den

beobachteten Symbolen $\mathbf{r}$ und unter Kenntnis des Kanals das *a posteriori* LLR $L(d_k \,|\, \mathbf{r}, \zeta(\mathbf{d}))$ des Datenbits d_k. Existiert zudem *a priori* Information $\zeta(d_k)$, wird diese mit berücksichtigt. Eine Implementierung dieser MAP-Entzerrung wird durch den Trellis-basierten MAP-Algorithmus[1] [BCJR74] ermöglicht, der als Metrik die euklidische Distanz verwendet.

Die zweite Komponente führt eine MAP-Decodierung der *a priori* Information $\zeta(c_k)$ mit Hilfe des senderseitig aufgeprägten Codes durch. Neben der Berechnung des *a posteriori* LLRs $L(c_k \,|\, \zeta(\mathbf{c}))$ wird auch eine harte Entscheidung bezüglich der Infobits $\hat{b}_k$ getroffen. Die Umsetzung der Decodierung kann abermals durch den MAP-Algorithmus vorgenommen werden, welcher in diesem Fall Übergangswahrscheinlichkeiten als Metrik verwendet.

Das Prinzip der Turbo-Entzerrung beruht nun auf dem iterativen Austausch der LLRs zwischen MAP-Entzerrer und MAP-Decoder. Es gelten dabei die für Turbo-Systeme essentiellen Voraussetzung [KST04]:

- **Voraussetzung 1:** Es darf nur extrinsische Information zwischen den SISO-Komponenten ausgetauscht werden.
- **Voraussetzung 2:** Die berechnete extrinsische Information einer Komponente dient als *a priori* Information für die andere Komponente.

Unter Berücksichtigung dieser Grundsätze ergibt sich das in Abbildung 4.1 dargestellte Blockschaltbild des Turbo-Entzerrers. Die zusätzliche Aufgabe des Deinterleavers, bzw. Interleavers, besteht in der Gewährleistung der für den Zusammenhang (3.23) nötigen Unabhängigkeit der Codebits bzw. Datenbits, die durch die starke örtliche Korrelation der Ausgänge beider Komponenten nicht zwangsläufig gegeben ist [KST04]. Für die weiteren Herleitungen wird folgende Notation verwendet

$$\begin{aligned} \lambda(\mathbf{c}) &= (\lambda(c_k))_{k=0}^{N_c-1} \\ \zeta(\mathbf{c}) &= (\zeta(c_k))_{k=0}^{N_c-1} \quad . \end{aligned} \tag{4.1}$$

[1] weitere Bezeichnungen: BCJR-Algorithmus, Forward-Backward-Algorithmus

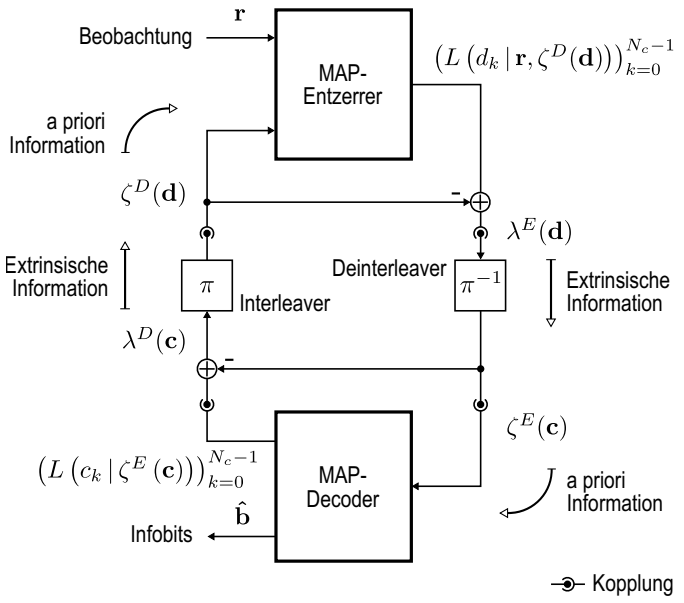


Abb. 4.1: Blockschaltbild des Turbo-Entzerrers

Gleiches gilt für die Datenbits b_k. Ein hochgestelltes E (Entzerrer) bzw. D (Decoder) kennzeichnet die Herkunft der Zuverlässigkeitsinformation.

Prinzipieller Ablauf Die Turbo-Entzerrung *einer* Beobachtung $\mathbf{r}$ läuft nun folgendermaßen ab:

↷ Für die erste MAP-Entzerrung liegt noch keine *a priori* Information vor und es wird mit $\zeta^D(\mathbf{d}) = \mathbf{0}$ das gleichwahrscheinliche Auftreten der Datenbits d_k angenommen. Diese Annahme ist durch die senderseitige Quellencodierung gerechtfertigt (s. Abs. 3.2).

↓ Unter Berücksichtigung von Voraussetzung 1 wird nach der Berechnung des *a posteriori* LLRs $L(d_k \,|\, \mathbf{r}, \mathbf{0})$ das extrinsische LLR $\lambda^E(d_k)$ durch Subtraktion des *a priori* LLRs $\zeta^D(d_k)$ bestimmt.

Da dies jedoch Null ist, entspricht die extrinsische Information in der *nullten* Iteration den *a posteriori* LLRs.

↓ Im Anschluss an das Deinterleaving werden die extrinsischen LLRs $\lambda^E(\mathbf{c})$ gemäß Voraussetzung 2 als *a priori* LLRs $\zeta^E(\mathbf{c})$ für die MAP-Decodierung verwendet.

↓ Der MAP-Decoder führt die Soft- und Hard-Decision durch und liefert eine Schätzung für die gesendeten Bits $\hat{\mathbf{b}}$ sowie die LLRs $\big(L(c_k \,|\, \zeta^E(\mathbf{c}))\big)_{k=0}^{N_c-1}$. Die nullte Turbo-Iteration ist abgeschlossen.

↑ Mit Beginn der *ersten* Turbo-Iteration unterscheidet sich der Ablauf von einem konventionellen Entzerrer. Für ein berechnetes *a posteriori* LLR gilt mit (3.23)

$$L(c_k \,|\, \zeta^E(\mathbf{c})) = \lambda^D(c_k) + \zeta^E(c_k) \,. \tag{4.2}$$

Durch diesen Zusammenhang kann das extrinsische LLR $\lambda^D(c_k)$ durch Subtraktion des *a priori* LLRs $\zeta^E(c_k)$ berechnet werden.

↑ Nach dem Interleaving dienen die extrinsischen LLRs $\lambda^D(\mathbf{d})$ als *a priori* LLRs $\zeta^E(\mathbf{d})$ für die zweite MAP-Entzerrung. Diese Information weicht im Idealfall von Null ab und spiegelt somit das *erlernte* Wissen über die gesendeten Datenbits d_k wider.

↓ Nach der erneuten MAP-Entzerrung und MAP-Decodierung ist die erste Turbo-Iteration abgeschlossen.

↻ Dieser Vorgang, das sukzessive Entzerren und Decodieren *einer* Beobachtung, wird bis zu einer gewissen Anzahl an Iterationen N_{it} fortgesetzt. Die Anzahl ist entweder statisch oder an eine Abbruchbedingung gekoppelt.

Der Ablauf des Verfahrens ist auch als Pseudo-Code in Listing 1 dargestellt. Die Analyse der Bitfehlerraten zeigt, dass mit jeder Turbo-Iteration die Zuverlässigkeit der Entscheidung über die Infobits $\hat{b}_k$ steigt[1]. Zusammenfassend stellt die Turbo-Entzerrung ein Verfahren dar, dessen Komplexität und Leistungsfähigkeit mit der Anzahl an Iterationen skaliert.

[1] Für den Fall, dass das Verfahren konvergiert (s. Abs. 4.4)

Pseudo-Code 1: Turbo-Entzerrung

Eingabe : Beobachtung $\mathbf{r}$
Ausgabe : Schätzung der gesendeten Infobit-Sequenz $\hat{\mathbf{b}}$

`// Initialisierung`
Allokiere L^D-Vektor und L^E-Vektor mit jeweils N_c Speichern
Initialisiere L^D-Vektor mit Nullen.

`// Turbo-Entzerrung`
Solange bis *die Abbruchbedingung erfüllt ist*
  Entzerrung: L^E = `MAP-Entzerrung` ($\mathbf{r}$, L^D)
  Extr. Information berechnen: $\mathrm{L}^E \leftarrow \mathrm{L}^E$ - L^D
  Deinterleaving: $\mathrm{L}^E \leftarrow \pi^{-1}(\mathrm{L}^E)$
  Decodierung: [L^D,$\hat{\mathbf{b}}$] = `MAP-Decodierung` (L^E)
  Extr. Information berechnen: $\mathrm{L}^D \leftarrow \mathrm{L}^D$ - L^E
  Interleaving: $\mathrm{L}^D \leftarrow \pi(\mathrm{L}^D)$
Ende
Ausgabe von $\hat{\mathbf{b}}$

4.2 SC-MMSE-FD Entzerrer

Obwohl die Komplexität der Turbo-Entzerrung für eine moderate Anzahl an Iterationen gegenüber der eines Optimalempfängers stark reduziert ist, zeigt sich bei näherer Betrachtung des MAP-Entzerrers, dass dessen Rechenaufwand exponentiell mit der Kanaleinflusslänge P wächst. Für stark dispersive Kanäle ist deshalb eine MAP-basierte Entzerrung nicht in Echtzeit umsetzbar. Im Folgenden wird das Soft Interference Cancellation Minimum Mean Square Error Frequency Domain (SC-MMSE-FD) Verfahren vorgestellt, das eine klassische, lineare Entzerrung unter Berücksichtigung der *a priori* LLRs durchführt und sich für Turbo-Systeme eignet.

4.2.1 Beschaltung

Die SC-MMSE-FD Entzerrung ist schematisch in Abbildung 4.2 dargestellt und wird nun isoliert vom Turbo-Entzerrer beschrieben. Im Kontext der Schätztheorie handelt es sich um einen MMSE-Schätzer

für die gesendeten Symbole, der auf das lineare Signalmodell (3.14) angewendet wird und über *a priori* Information verfügt.

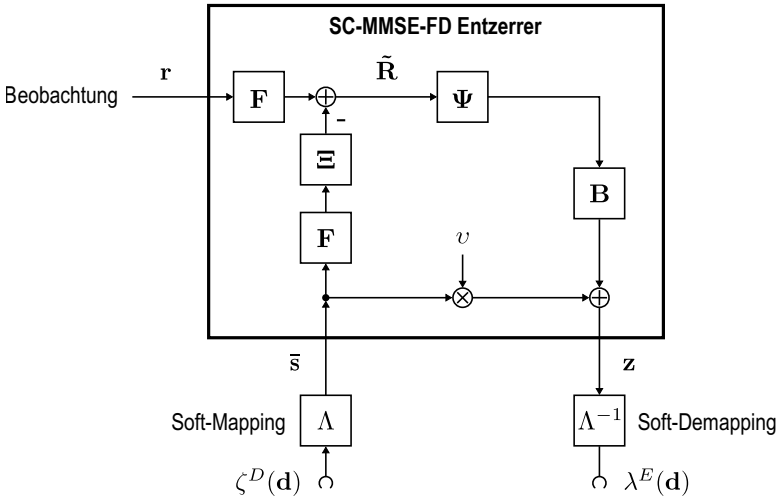


Abb. 4.2: Blockschaltbild des SC-MMSE-FD Entzerrers

Die Eingangssignale sind sowohl die empfangenen Symbole $\mathbf{r}$ als auch *a priori* Information in Form *geschätzter* Sendesymbole $\bar{\mathbf{s}} \in \mathbb{C}^{N_s}$. Die Schätzung wird durch das Abbilden der LLRs $\zeta^D(\mathbf{d})$ auf Punkte im Signalraumdiagramm bestimmt

$$\bar{\mathbf{s}} = \mathrm{E}\left\{\mathbf{s} \,|\, \zeta^D(\mathbf{d})\right\} . \tag{4.3}$$

Für perfekte *a priori* Information entsprechen die Punkte den Symbolen des Modulationsalphabets $\mathcal{S}$. Ohne Vorkenntnis, meist in der nullten Iteration, ist $\bar{\mathbf{s}}$ der Nullvektor. Das *Soft-Mapping* wird detaillierter in Abschnitt 4.2.4 für die verwendeten digitalen Modulationsverfahren erläutert. Am Ausgang liefert der SC-MMSE-FD Entzerrer die entzerrten Symbole $\mathbf{z}$.

4.2.2 Herleitung

Bei der Konstruktion eines Entzerrers für einen Turbo-Entzerrer, der neben den empfangenen Symbolen zusätzlich *a priori* Information über die gesendeten Symbole nutzt, gilt es erneut Voraussetzung 1 zu berücksichtigen: Damit lediglich extrinsische Information nach der Entzerrung vorliegt, darf zu keinem Zeitpunkt k ein funktioneller Zusammenhang zwischen einem entzerrten Symbol z_k und einem geschätztem Symbol $\bar{s}_k$ am Eingang bestehen. Mit dem Soft Interference Cancellation Ansatz nach [TKS02] und einer Entzerrermatrix $\mathbf{G} \in \mathbb{C}^{N_s \times N_s}$ im Zeitbereich folgt für die entzerrten Symbole

$$\mathbf{z} = \boldsymbol{\Upsilon}\bar{\mathbf{s}} + \mathbf{G}(\mathbf{r} - \mathbf{H}\bar{\mathbf{s}}) \; . \tag{4.4}$$

Wird der Vorfaktor als $\boldsymbol{\Upsilon} = \mathrm{diag}\,(\mathbf{GH})$ gewählt, ist Voraussetzung 1 erfüllt. Anschaulich kann SC als ein Verfahren betrachtet werden, das nach der Entzerrung des Empfangssignals durch $\mathbf{Gr}$, die verbleibende Intersymbolinterferenz durch die Schätzung $(\mathbf{GH} - \boldsymbol{\Upsilon})\bar{\mathbf{s}}$ unterdrückt.

Im nächsten Schritt wird die Entzerrermatrix $\mathbf{G}$ durch das Minimieren der mittleren Fehlerleistung zwischen den gesendeten Symbolen $\mathbf{s}$ und den entzerrten Symbolen $\mathbf{z}$ ermittelt. Zu berücksichtigen ist dabei, dass der Vorfaktor $\boldsymbol{\Upsilon}$ zu einer Skalierung des Ausgangssignals führt. Das mathematische Optimierungsproblem zur Bestimmung der Entzerrermatrix lautet demnach

$$\mathbf{G} = \arg\min_{\mathbf{G}} \mathrm{E}\left\{\|\boldsymbol{\Upsilon}\mathbf{s} - \mathbf{z}\|^2\right\} \; . \tag{4.5}$$

Das Lösen dieses Problems kann durch die Tatsache, dass $\boldsymbol{\Upsilon}$ erst aus der Entzerrermatrix $\mathbf{G}$ resultiert, für die Optimierung vereinfacht werden. Mithilfe des Orthogonalitätsprinzips [Kay93] lautet der Lösungsansatz

$$\mathrm{E}\left\{(\mathbf{s} - \mathbf{G}(\mathbf{r} - \mathbf{H}\bar{\mathbf{s}}))\,\mathbf{r}^H\right\} \stackrel{!}{=} \mathbf{0} \; . \tag{4.6}$$

Unter Berücksichtigung des Signalmodells (3.14) und mit der Annahme, dass das Interleaving die Unabhängigkeit der geschätzten Symbole $\bar{s}_k$ garantiert, folgt als Lösung für das Optimierungsproblem (4.5)

$$\mathbf{G} = \mathbf{H}^H \left((1-\varphi)\mathbf{H}\mathbf{H}^H + \sigma_0^2\mathbf{I}\right)^{-1} \; , \tag{4.7}$$

wobei die mittlere Leistung der geschätzten Symbole $\varphi = \mathrm{E}\{\|\bar{\mathbf{s}}\|^2\}$ durch das arithmetische Mittel angenähert wird

$$\varphi = \frac{1}{N_s} \|\bar{\mathbf{s}}\|^2 \,. \tag{4.8}$$

Da die Kanalfaltungsmatrix $\mathbf{H}$ aufgrund der Konstruktion des Sendesignals eine zyklische Form besitzt, wird diese durch die Fourier-Matrizen aus (3.1) und (3.2) diagonalisiert

$$\mathbf{H} = \mathbf{B}\boldsymbol{\Xi}\mathbf{F} \,. \tag{4.9}$$

Die Diagonalmatrix $\boldsymbol{\Xi}$ entspricht der Kanalübertragungsmatrix.

Nach dem Einsetzen von (4.7) und (4.9) in (4.4) berechnen sich die entzerrten Symbole am Ausgang des SC-MMSE-FD Entzerrers gemäß

$$\begin{aligned} \mathbf{z} &= \upsilon\bar{\mathbf{s}} + \mathbf{B}\underbrace{\boldsymbol{\Xi}\left((1-\varphi)\,\boldsymbol{\Xi}\boldsymbol{\Xi}^H + \sigma_0^2\mathbf{I}\right)^{-1}}_{\boldsymbol{\Psi}} \cdot \left(\mathbf{F}\mathbf{r} - \boldsymbol{\Xi}\mathbf{F}\bar{\mathbf{s}}\right) \\ &= \upsilon\bar{\mathbf{s}} + \mathbf{B}\boldsymbol{\Psi}\left(\mathbf{F}\mathbf{r} - \boldsymbol{\Xi}\mathbf{F}\bar{\mathbf{s}}\right) . \end{aligned} \tag{4.10}$$

Die Diagonalmatrix $\boldsymbol{\Psi}$ dient zur Entzerrung des residualen Signals im Frequenzbereich. Für den Vorfaktor folgt aus (4.9) entsprechend $\boldsymbol{\Upsilon} = \upsilon\mathbf{I}$, mit

$$\upsilon = \frac{1}{N_s} \operatorname{spur}(\boldsymbol{\Psi}\boldsymbol{\Xi}) \,. \tag{4.11}$$

Das SC-MMSE-FD Verfahren ist als Pseudo-Code in Listing 2 wiedergegeben.

4.2.3 Äquivalentes AWGN-Modell

Der Einfluss des SC-MMSE-FD Entzerrers und des Kanals auf die Modulationssymbole s_k kann durch ein vereinfachtes AWGN-Modell der Form

$$z_k \approx \upsilon s_k + w_k \tag{4.12}$$

Pseudo-Code 2: SC-MMSE-FD Entzerrer mit Soft-(De)Mapping

Eingabe : Beobachtung $\mathbf{r}$, LLRs $\zeta^D(\mathbf{d})$, Modulation ($\mathcal{S}$,$\mathcal{C}$), Kanalimpulsantwort $\mathbf{h}$, Rauschleistung σ_0^2
Ausgabe : LLRs $\lambda^E(\mathbf{d})$

`// Initialisierung`
Allokiere $\mathbf{\Psi}$-Vektor, $\bar{\mathbf{s}}$-Vektor, $\bar{\mathbf{S}}$-Vektor, $\mathbf{\Xi}$-Vektor, $\mathbf{R}$-Vektor, $\mathbf{z}$-Vektor und $\mathbf{Z}$-Vektor mit jeweils N_s Speicherelementen.
Initialisiere $\mathbf{\Xi}$ = `DFT` ($\mathbf{h}$, Zero-Padding)
Initialisiere $\mathbf{R}$ = `DFT` ($\mathbf{r}$)

`// Soft-Mapping`
Berechne gemäß (4.16): $\bar{\mathbf{s}}$ = `Soft-Mapping` $\left(\zeta^D(\mathbf{d})\right)$

`// SC-MMSE-FD Entzerrer`
Berechne mittlere Leistung φ der Symbole $\bar{\mathbf{s}}$ gemäß (4.8)
Berechne Übertragungsfunktion $\mathbf{\Psi}$ gemäß (4.10)
Bestimme Verteilungsparameter υ und σ_z^2 gemäß (4.11) und (4.14)
Berechne DFT: $\bar{\mathbf{S}}$ = `DFT` ($\bar{\mathbf{s}}$)
SC-MMSE-FD (elementw. Mult.: $\odot$): $\mathbf{Z} = \mathbf{\Psi} \odot (\mathbf{R}$ - $\mathbf{\Xi} \odot \bar{\mathbf{S}})$
Rücktransformation: $\mathbf{z} = \upsilon\bar{\mathbf{s}}$ + `IDFT` ($\mathbf{Z}$)

`// Soft-Demapping`
Berechne gemäß (4.18): $\lambda^E(\mathbf{d})$ = `Soft-Demapping` $\left(\mathbf{z}, \upsilon, \sigma_z^2, \zeta^D(\mathbf{d})\right)$

angenähert werden [KM07]. Der Vorfaktor υ entsteht dabei durch die Skalierung (4.11) im Entzerrer. Die verbleibende Intersymbolinterferenz in den entzerrten Symbolen z_k wird durch ein weißes Gauß'sches Rauschen w_k approximiert. Zur Bestimmung der Charakteristik dieses Rauschens werden sowohl der Erwartungswert $\mathrm{E}\{w_k\}$ sowie die Varianz $\mathrm{Var}\{w_k\}$ berechnet. Für ersteres gilt

$$\begin{aligned} \mathrm{E}\{w_k\} &= \mathrm{E}\{z_k - \upsilon s_k\} \\ &= 0\,. \end{aligned} \tag{4.13}$$

Entsprechend folgt für die Varianz mit (4.10)

$$\begin{aligned} \mathrm{Var}\{w_k\} &= \mathrm{Var}\{z_k - \upsilon s_k\} \\ &= \mathrm{E}\{(z_k - \upsilon s_k)^2\} \\ &= \upsilon^2(\varphi - 1) + \upsilon\,. \end{aligned} \tag{4.14}$$

Die Verteilung des Rauschens entspricht somit $\mathcal{CN}\left(0, \upsilon^2(\varphi - 1) + \upsilon\right)$. Das Modell dient zur vereinfachten Analyse der Leistungsfähigkeit des SC-MMSE-FD Entzerrers und wird zum Soft-Demapping benötigt. Letzteres bildet die entzerrten Symbole auf LLRs ab und ist in Abschnitt 4.2.5 beschrieben.

4.2.4 Soft-Mapping

Bei Verwendung des SC-MMSE-FD Entzerrers ist es nötig, eine Abbildung der entzerrten Symbole z_k auf die LLRs $\lambda^E(d_k)$, bzw. der *a priori* Information $\zeta^D(d_k)$ auf die Symbole $\bar{s}_k$ durchzuführen. Der mit Soft-Demapping und Soft-Mapping bezeichnete Vorgang entspricht dabei einer Transformation der Entscheidungszuverlässigkeit zwischen Bit- und Symboldomäne und ist in Abbildung 4.2 durch Λ bzw Λ^{-1} gekennzeichnet. Da beide Vorgänge zeitunabhängig sind, kann der Übersichtlichkeit halber in diesem Abschnitt auf das Mitführen des Zeitindexes k verzichtet werden. Die Betrachtungen finden lediglich auf ein Symbol bzw. auf eine Gruppe von LLRs bezogen statt.

Unter Kenntnis des Modulationsalphabets $\mathcal{S}$ und der Codierungsvorschrift $\mathcal{C}$ bildet das Soft-Mapping Λ die LLRs von $\mathrm{ld}(M)$ Datenbits auf eine Schätzung für das gesendete Symbol ab

$$\Lambda\colon \mathbb{R}^{\mathrm{ld}(M)} \to \mathbb{C}, \quad \left\{\zeta^D(d_i)\right\}_{i=0}^{\mathrm{ld}(M)-1} \mapsto \bar{s}\,. \tag{4.15}$$

Mithilfe von (4.3) und der Annahme, dass der Interleaver die Unabhängigkeit der einzelnen LLRs am Eingang sicherstellt, folgt

$$\begin{aligned}
\bar{s} &= \mathrm{E}\left\{ s \,\middle|\, \left\{\zeta^D(d_i)\right\}_{i=0}^{\mathrm{ld}(M)-1} \right\} \\
&= \sum_{m=0}^{M-1} S_m \cdot P\left(s = S_m \,|\, \zeta^D(d_0), \ldots, \zeta^D(d_{\mathrm{ld}(M)-1})\right) \\
&= \sum_{m=0}^{M-1} S_m \prod_{i=0}^{\mathrm{ld}(M)-1} P\left(d_i = d_{m,i} \,|\, \zeta^D(d_i)\right) \\
&= \sum_{m=0}^{M-1} S_m \prod_{i=0}^{\mathrm{ld}(M)-1} \frac{1}{1 + e^{(2d_{m,i}-1)\zeta^D(d_i)}}\,.
\end{aligned} \tag{4.16}$$

Die Summe der Modulationssymbole $S_m \in \mathcal{S}$, gewichtet mit ihrer Auftretenswahrscheinlichkeit, ergibt die Schätzung für ein *a priori* Symbol $\bar{s}$. Die Wahrscheinlichkeit der Bitgruppe $\{d_{m,i}\}_{i=0}^{\mathrm{ld}(M)-1}$ eines Modulationssymbols S_m wird dabei durch die LLRs $\zeta^D(d_i)$ bestimmt. Der letzte Schritt in (4.16) ergibt sich aus dem allgemeinen Zusammenhang [HLY02], dass sich die Berechnungen von $P(x=1) = (1+e^{-L(x)})^{-1}$ und $P(x=0) = (1+e^{L(x)})^{-1}$ lediglich in einem Vorzeichen unterscheiden und dieses durch $2x-1$ bestimmt werden kann.

4.2.5 Soft-Demapping

Das Soft-Demapping stellt die Umkehroperation zum Soft-Mapping dar und bildet ein entzerrtes Symbol z auf $\mathrm{ld}(M)$ LLRs ab

$$\Lambda^{-1}\colon \mathbb{C} \to \mathbb{R}^{\mathrm{ld}(M)}, \quad z \mapsto \left\{\lambda^E(d_i)\right\}_{i=0}^{\mathrm{ld}(M)-1} . \tag{4.17}$$

Da der SC-MMSE-FD Entzerrer bereits mit dem Ziel entworfen wurde rein extrinsische Information zu berechnen, lautet die Abbildungsvorschrift unter Verwendung von (3.23) [TS11]

$$\lambda^E(d_i) = \ln\left(\frac{\sum\limits_{\substack{S_m:\\ d_{m,i}=1}} p(z\,|\,S_m)\prod_{l=0,l\neq i}^{\mathrm{ld}(M)-1} P\big(d_l = d_{m,l}\,|\,\zeta^D(d_l)\big)}{\sum\limits_{\substack{S_m:\\ d_{m,i}=0}} p(z\,|\,S_m)\prod_{l=0,l\neq i}^{\mathrm{ld}(M)-1} P\big(d_l = d_{m,l}\,|\,\zeta^D(d_l)\big)}\right), \tag{4.18}$$

wobei $d_{m,i}$ das i-te Bit der Codierung $\mathcal{C}$ des m-ten Modulationssymbols $S_m \in \mathcal{S}$ darstellt (s. Abb. 3.4). Die bedingte Verteilungsdichte $p(z\,|\,S_m)$ eines entzerrten Symbols z kann, wie bereits in Abschnitt 4.2.3 beschrieben, durch ein weißes Gauß'sches Rauschen mit der Varianz $\sigma_z^2 = \upsilon^2(\varphi-1)+\upsilon$ angenähert werden

$$p(z\,|\,S_m) = \frac{1}{\pi\sigma_z^2}\cdot\exp\left(-\frac{1}{\sigma_z^2}|z-\upsilon S_m|^2\right). \tag{4.19}$$

Für binäre Modulationsverfahren, wie z.B. BPSK oder auch QPSK[1], entfällt das Produkt in (4.18) sowohl im Zähler als auch im Nenner.

[1] Bei QPSK können der Imaginär- und Quadraturteil als zwei unabhängige BPSK-Komponenten betrachtet werden.

Für höherstufige Verfahren ist die *a priori* Information $\zeta^D(d_i)$ zur Berechnung der Auftretenswahrscheinlichkeit der Bitgruppen zu berücksichtigen. Dies führt zu einer verbesserten Leistungsfähigkeit im Bereich von hohen SNR pro Bit.

4.3 MAP-Decoder

Der folgende Abschnitt widmet sich dem Fehlerschutz im Sendesignal und beschreibt die Umsetzung des SISO-Decoders, der zweiten Komponente des Turbo-Entzerrers. Der bereits mehrfach erwähnte MAP-Algorithmus ist ein universelles Verfahren zur Berechnung von *a posteriori* Wahrscheinlichkeiten und wurde erstmals 1974 von Bahl, Cocke, Jelinek und Raviv[1] [BCJR74] zur Decodierung von linearen Codes vorgeschlagen. Auf Basis einer verrauschten Beobachtung und vorhandenem *a priori* Wissen lassen sich sowohl die Zustands- als auch Übergangswahrscheinlichkeiten im Trellis-Diagramm berechnen und die Soft- und Hard-Decision durchführen. Dieser MAP-Decoder ist im Sinne der minimalen Symbolfehlerwahrscheinlichkeit optimal [BCJR74].

Wie in Abschnitt 4.2 erläutert, ist die Entzerrung des Empfangssignals mithilfe des MAP-Algorithmus aufgrund der Komplexität nicht in Echtzeit umsetzbar und wurde deshalb nicht weiter verfolgt. Demgegenüber ist die Komplexität des MAP-Algorithmus zur Decodierung allein vom Faltungscode $\mathbf{C}$ abhängig. Diese Eigenschaft bietet somit Spielraum zur senderseitigen Dimensionierung oder empfängerseitigen Optimierung.

Durch die Einführung des Universal Mobile Telecommunications Systems (UMTS), das als erster Funkstandard Turbo-Codes zur Fehlerkorrektur einsetzt, wurde die Entwicklung von applikations- und prozessorspezifischen Modifikationen des MAP-Algorithmus stark vorangetrieben. Die Beschreibungen und Herleitungen in dieser Arbeit beschränken sich zunächst auf die grundlegenden Abläufe des Algorithmus. Anschließend wird auf die fundamentalen Implementierungsa-

[1] Das Akronym *BCJR* dient üblicherweise in der Literatur zur Bezeichnung des Algorithmus.

spekte eingegangen und die Modifikation des Max*-Log-MAP Decoders erläutert.

Zur SISO-Decodierung kann ebenso der Soft-Output Viterbi Algorithm (SOVA) verwendet werden [Vit67][HH89]. Ein Vergleich der Komplexität zeigt, dass die Anzahl der nötigen Operationen für die Implementierung des ML-basierten SOVAs in etwa der Hälfte der Operationen eines MAP-Decoders[1] entspricht [RVH95]. Diese Aufwandsreduzierung wird allerdings auf Kosten eines um 0,6 dB schlechteren SNR pro Bit erkauft [EPG94]. Darüber hinaus zeigt der MAP-Decoder ein größeres Potential hinsichtlich einer parallelisierten Implementierung (s. Kap. 6). Aus diesen Gründen wird der MAP-Decoder, genauer gesagt die Max*-Log-MAP Variante, zur SISO-Decodierung in dieser Arbeit verwendet.

Nach der Einführung der verwendeten Notation und des Trellis-Diagramms in Abschnitt 4.3.1 wird die Beschaltung des MAP-Decoders im Turbo-Entzerrer in Abschnitt 4.3.2 beschrieben. Der prinzipielle Ablauf wird anschließend für SCCs in Abschnitt 4.3.3 erläutert. Abschnitt 4.3.4 beschäftigt sich mit den nötigen Implementierungsaspekten und dem Max*-Log-MAP Decoder. Die Modifikationen zur Decodierung von SCCCs werden in Abschnitt 4.3.5 hervorgehoben.

4.3.1 Notation

Die Decodierung eines Faltungscodes $\mathbf{C}$ kann grafisch durch ein Trellis-Diagramm dargestellt werden. Abbildung 4.3 zeigt die im Folgenden verwendete Notation zur Bezeichnung von Zuständen und Übergängen. Wie bereits in Abschnitt 3.2.2 erwähnt, beschränkt sich die Beschreibung des Decoders auf Faltungscodes mit Coderate $r = \frac{1}{n_c}$. Mit einem Übergang im Trellis sind demnach $n_b = 1$ Infobit und n_c Codebits verknüpft. Die Anzahl der Zustände hängt von der Länge des Schieberegisters im Encoder ab und beträgt 2^ν. Die Länge des Trellis wird durch die Länge der Infobit-Sequenz bestimmt. Ein Pfad durch den Trellis besteht aus N_b Übergängen und ist mit insgesamt N_b Infobits bzw. N_c Codebits verknüpft.

[1] Im Fall der Max*-Log-MAP Variante

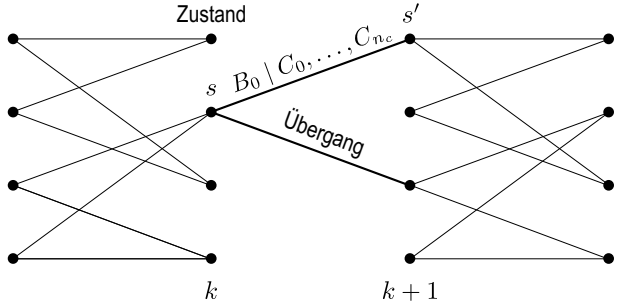


Abb. 4.3: Trellis-Diagramm

4.3.2 Beschaltung

Die Eingaben des MAP-Decoders (s. Abb. 4.4) bilden die *a priori* LLRs $L^{\mathrm{i}}(\mathbf{c}) = \zeta^E(\mathbf{c}) \in \mathbb{R}^{N_c}$, die den Zuverlässigkeiten der Codebits entsprechen. Mit der Annahme, dass der Deinterleaver die Unabhängigkeit der LLRs sicherstellt, ist die Voraussetzung zur MAP-Decodierung erfüllt [TS11]. Da im Gegensatz zur Turbo-Decodierung [BGT93] kein *a priori* Wissen über die Wahrscheinlichkeit der Infobits existiert, wird dieser Eingang des MAP-Decoders mit $L^{\mathrm{i}}(\mathbf{b}) = \mathbf{0}$ belegt.

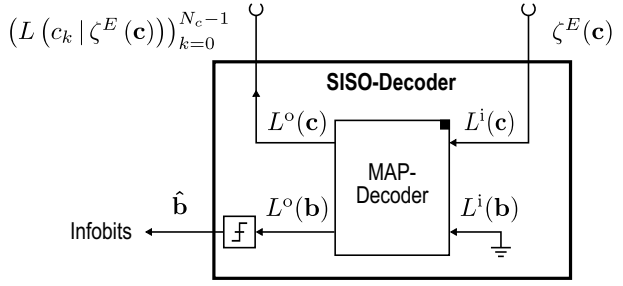


Abb. 4.4: Integration des MAP-Decoders zur Decodierung von SCCs

Der Ausgang des Decoders sind die geschätzten Infobits $\hat{\mathbf{b}}$ sowie die *a posteriori* LLRs $L^{\mathrm{o}}(\mathbf{c}) = L\left(c_k \mid L^{\mathrm{i}}(\mathbf{c})\right)_{k=0}^{N_c-1}$. Ersteres wird durch eine harte Entscheidung bezüglich der LLRs $L^{\mathrm{o}}(\mathbf{b})$ bestimmt (s. Abs. 3.3.2).

4.3.3 Funktionsweise

Für das *a posteriori* LLR eines Codebits am Ausgang des MAP-Decoders gilt

$$L^{\mathrm{o}}(c_k) = \ln\left(\frac{P(c_k = 1 \,|\, L^{\mathrm{i}}(\mathbf{c}))}{P(c_k = 0 \,|\, L^{\mathrm{i}}(\mathbf{c}))}\right) . \tag{4.20}$$

Im Weiteren ist zu berücksichtigen, dass aufgrund der Coderate eine Transformation zwischen den Indizes der Codebits und der Zustände nötig ist. Unter Berücksichtigung des Trellis-Diagramms in Abbildung 4.3 kann (4.20) umformuliert werden. Die Wahrscheinlichkeit, dass zum Zeitpunkt

$$k' = n_c k + \kappa_c, \quad \kappa_c = 0, \ldots, n_c - 1 \tag{4.21}$$

das Codebit $c_{k'}$ einer 1 entspricht, ist dabei mit der Summe der Wahrscheinlichkeiten aller Übergänge gleichzusetzen, die zu dieser 1 führen können. Demzufolge müssen diejenigen Übergänge von s nach s' ausgewertet werden, die an der κ_c-ten Stelle das Codebit $C_{\kappa_c} = 1$ aufweisen. Selbiges gilt für $C_{\kappa_c} = 0$ und führt schließlich auf

$$L^{\mathrm{o}}(c_{k'}) = \ln\left(\frac{\sum\limits_{\substack{s \to s' \\ C_{\kappa_c}=1}} P(s, s' \,|\, L^{\mathrm{i}}(\mathbf{c}))}{\sum\limits_{\substack{s \to s' \\ C_{\kappa_c}=0}} P(s, s' \,|\, L^{\mathrm{i}}(\mathbf{c}))}\right) . \tag{4.22}$$

Die bedingte Wahrscheinlichkeit in Zähler und Nenner ist durch den Satz von Bayes [JW02] in die Verbunddichte überführbar

$$p(s, s', L^{\mathrm{i}}(\mathbf{c})) = P(s, s' \,|\, L^{\mathrm{i}}(\mathbf{c})) \cdot p(L^{\mathrm{i}}(\mathbf{c})) . \tag{4.23}$$

Da $p(L^{\mathrm{i}}(\mathbf{c}))$ unabhängig vom Zustandsübergang ist, entfällt dieser Term nach Einsetzen von (4.23) in (4.22). Die Anwendung der Kettenregel [Gut10] ermöglicht es die Verbundwahrscheinlichkeit in drei unabhängige Terme aufzuspalten, die anschaulich die Vergangenheit, die Gegenwart und die Zukunft eines Pfades durch den Trellis widerspiegeln

$$p(s, s', L^{\mathrm{i}}(\mathbf{c})) = \alpha_k(s) \cdot \gamma_k(s, s') \cdot \beta_{k+1}(s') . \tag{4.24}$$

Der erste Term $\alpha_k(s)$ gibt die Wahrscheinlichkeit eines Zustands s zum Zeitpunkt k an und wird aus den LLRs $L^{\mathrm{i}}(\mathbf{c})$ bis zu diesem Zeitpunkt bestimmt

$$\alpha_k(s) = p\left(s, \left(L^{\mathrm{i}}(c_l)\right)_{l=0}^{n_c k-1}\right) . \tag{4.25}$$

Äquivalent gilt für die Zustandswahrscheinlichkeit $\beta_{k+1}(s')$, dass die Wahrscheinlichkeit für die zukünftigen LLRs aufgrund der Markov-Eigenschaft [JW02] lediglich vom Zustand s' zum Zeitpunkt $k+1$ abhängt

$$\beta_{k+1}(s') = p\left(\left(L^{\mathrm{i}}(c_l)\right)_{l=n_c(k+1)}^{N_c-1} \;\middle|\; s'\right) . \tag{4.26}$$

Der mittlere Term $\gamma_k(s, s')$ gibt schließlich die Übergangswahrscheinlichkeit von Zustand s nach s' an und bildet somit die Verknüpfung zwischen Vergangenheit und Zukunft. Erneut folgt mithilfe der Markov-Eigenschaft, dass die Wahrscheinlichkeit für den Folgezustand lediglich vom Ausgangszustand und den LLRs während des Übergangs abhängt

$$\gamma_k(s, s') = p\left(s', \left(L^{\mathrm{i}}(c_l)\right)_{l=n_c k}^{n_c(k+1)-1} \;\middle|\; s\right) . \tag{4.27}$$

Die Anwendung der Kettenregel liefert schließlich eine berechenbare Form zur Bestimmung der Übergangswahrscheinlichkeiten (Metrik)

$$\gamma_k(s, s') = \frac{1}{n_b}\, p\left(\left(L^{\mathrm{i}}(c_l)\right)_{l=n_c k}^{n_c(k+1)-1} \;\middle|\; (C_{\kappa_c})_{\kappa_c=0}^{n_c-1}\right) . \tag{4.28}$$

Der Vorfaktor ergibt sich aus der Tatsache, dass aufgrund des fehlenden *a priori* Wissens $L^{\mathrm{i}}(\mathbf{b})$ über die Wahrscheinlichkeit der Infobits, alle möglichen Kombinationen während des Übergangs von s nach s' als gleichwahrscheinlich gelten. Die bedingte Wahrscheinlichkeitsdichte lässt sich nun für jeden Trellis-Übergang durch die LLRs $L^{\mathrm{i}}(\mathbf{c})$ und die mit diesem Übergang verknüpften Codebits $(C_{\kappa_c})_{\kappa_c=0}^{n_c-1}$ bestimmen. Für die Metrik folgt demnach

$$\gamma_k(s, s') = \frac{1}{n_b} \prod_{\kappa_c=0}^{n_c-1} \left(1 + \exp\left((2C_{\kappa_c} - 1) \cdot L^{\mathrm{i}}(c_{n_c k+\kappa_c})\right)\right)^{-1} . \tag{4.29}$$

Ist aufgrund der Struktur des Faltungscodes ein Übergang von s nach s' nicht möglich, gilt $\gamma_k(s, s') = 0$.

Die Berechnung der Zustandswahrscheinlichkeiten lässt sich mithilfe der Übergangswahrscheinlichkeiten rekursiv durchführen. Es folgt die für den BCJR bekannte *Forward-Recursion*

$$\alpha_k(s') = \sum_{\forall s} \gamma_k(s, s') \cdot \alpha_{k-1}(s) \tag{4.30}$$

bzw. *Backward-Recursion*

$$\beta_k(s) = \sum_{\forall s'} \gamma_k(s, s') \cdot \beta_{k+1}(s') \; . \tag{4.31}$$

Die Rekursionen sind möglich, da sich der Encoder sowohl zu Beginn als auch zum Ende einer Infobit-Sequenz in einem definierten Zustand befindet. Als Startzustand wird ohne Beschränkung der Allgemeinheit der Nullzustand angenommen. In diesem Fall sind die Speicher des Encoders mit 0 belegt. Ebenso wird der Encoder zum Ende hin wieder in den Nullzustand terminiert (s. Abs. 3.2.2).

Mithilfe der Wahrscheinlichkeiten (4.29), (4.30) und (4.31) folgt die Ausgangsformel (4.22) zur Bestimmung der *a posteriori* Information zu

$$L^{\mathrm{o}}(c_{k'}) = \ln \left(\frac{\sum\limits_{\substack{s \to s' \\ C_{\kappa_c}=1}} \alpha_k(s) \cdot \gamma_k(s, s') \cdot \beta_{k+1}(s')}{\sum\limits_{\substack{s \to s' \\ C_{\kappa_c}=0}} \alpha_k(s) \cdot \gamma_k(s, s') \cdot \beta_{k+1}(s')} \right) . \tag{4.32}$$

Die Berechnung der LLRs $L^{\mathrm{o}}(\mathbf{b})$ verläuft ähnlich zu der der Codebits. Die Wahrscheinlichkeit, dass zum Zeitpunkt k das Infobit b_k einer 1 bzw. 0 entsprach, ist gleichzusetzen mit der Wahrscheinlichkeit aller Übergänge von s nach s', die durch $B_0 = 1$ bzw. $B_0 = 0$ initiiert werden

$$L^{\mathrm{o}}(b_k) = \ln \left(\frac{\sum\limits_{\substack{s \to s' \\ B_0=1}} \alpha_k(s) \cdot \gamma_k(s, s') \cdot \beta_{k+1}(s')}{\sum\limits_{\substack{s \to s' \\ B_0=0}} \alpha_k(s) \cdot \gamma_k(s, s') \cdot \beta_{k+1}(s')} \right) . \tag{4.33}$$

Das Treffen einer harten Entscheidung liefert schließlich die Schätzung für die gesendeten Infobits

$$\hat{b}_k = \begin{cases} 1, & L^{\mathrm{o}}(b_k) \geq 0 \\ 0, & L^{\mathrm{o}}(b_k) < 0 \end{cases} . \tag{4.34}$$

Der prinzipielle Ablauf des Algorithmus ist als Pseudo-Code in Listing 3 wiedergegeben.

Pseudo-Code 3: SISO-Decoder

Eingabe : $\zeta^E(\mathbf{c})$
Ausgabe : $\left(L\left(c_k \,|\, \zeta^E(\mathbf{c})\right)\right)_{k=0}^{N_c-1}$, $\hat{\mathbf{b}}$

`// Initialisierung`
Allokiere γ-Matrix mit $\left[2^{1+\nu} \times N_b\right]$, α-Matrix und β-Matrix mit jeweils $[2^{\nu} \times (1 + N_b)]$ Speicherelementen
Initialisiere erste Spalte der α-Matrix mit Null bzw. Nullzustand mit 1
Initialisiere letzte Spalte der β-Matrix mit Null bzw. Nullzustand mit 1

`// Bestimmung der Übergangsw'keiten`
Für jeden *Übergang im Trellis*
  Berechne Übergangsw'keit gemäß (4.29)
Ende

`// Forward- und Backward-Recursion`
Von *Anfang bis Ende des Trellis*
  Für jeden *Folgezustand*
    Berechne die W'keit eines Folgezustands gemäß (4.30)
  Ende
Ende
Von *Ende bis Anfang des Trellis*
  Für jeden *Vorgängerzustand*
    Berechne die W'keit eines Vorgängerzustand gemäß (4.31)
  Ende
Ende

`// Bestimmung der Ausgaben`
Für jeden *Trellis-Schritt k*
  Berechne $L\left(\cdot \,|\, \zeta^E(\mathbf{c})\right)$
  Berechne b_k gemäß (4.34)
Ende

4.3.4 Max*-Log-MAP Decoder

Bei der Implementierung des MAP-Decoders treten numerische Probleme auf [Pet94], die sich durch gezielte Modifizierungen des Algorithmus vermeiden lassen. Die aus Sicht der Numerik unabdingbare Modifikation ist die Durchführung des MAP-Decoders im Logarithmischen. Der Grund hierfür liegt in der numerischen Instabilität, die sich durch die iterative Berechnung von Wahrscheinlichkeiten (Werte ≤ 1) ergibt. Zur Reduzierung des notwendigen Dynamikbereichs der Zahlendarstellung wird deshalb eine Logarithmierung aller Berechnungen vorgenommen. Die Multiplikation von Wahrscheinlichkeiten vereinfacht sich somit zur Addition ihrer Logarithmen. Im Falle des Max*-Log-MAP Decoders [RVH95] werden zudem die Operationen $\ln(e^a + e^b)$ durch eine modifizierte Maximum-Operation angenähert

$$\ln(e^a + e^b) \approx \max(a,b) + \ln\left(1 + e^{-|a-b|}\right) =: \max^*(a,b)\ . \tag{4.35}$$

Durch den Korrekturterm ist der Verlust an SNR pro Bit gegenüber dem optimalen MAP-Decoder vernachlässigbar [RVH95]. Aufgrund der Vielzahl dieser Operationen reduziert sich jedoch der Implementierungsaufwand signifikant, da auf die rechenintensive Funktionen $\exp(\cdot)$ und $\ln(\cdot)$ prinzipiell verzichtet werden kann. Die Werte des Korrekturterms können z.B. in einer Lookup-Table hinterlegt werden.

Mithilfe der logarithmischen Darstellung (gekennzeichnet durch ein Unterstrich) und der Approximation (4.35) folgt für die Berechnung der Forward- und Backward-Recursion

$$\begin{aligned} \underline{\alpha}_k(s') &= \max_{\forall s}^{*} \left(\underline{\gamma}_k(s,s') + \underline{\alpha}_{k-1}(s)\right)\ , \text{bzw.} \\ \underline{\beta}_k(s) &= \max_{\forall s'}^{*} \left(\underline{\gamma}_k(s,s') + \underline{\beta}_{k+1}(s')\right)\ . \end{aligned} \tag{4.36}$$

Die $\max^*$-Operation bezieht sich in diesen Fällen auf alle Vorgängerzustände von s' (Forward-Recursion) bzw. von s (Backward-Recursion). Abbildung 4.5 zeigt exemplarisch den Ablauf der Forward-Recursion.

Nach den Rekursionen kann die Soft-Decision zur Berechnung der a

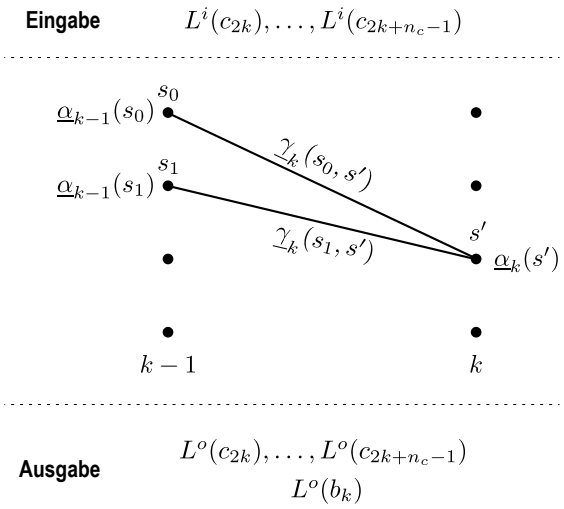


Abb. 4.5: Forward-Recursion des Max*-Log-MAP Decoders

posteriori LLRs unter Beachtung der Indizierung (4.21) gemäß

$$\begin{aligned} L^o(c_{k'}) = & \max_{\substack{s \to s' \\ C_{\kappa_c}=1}}{}^{*} \left(\underline{\alpha}_k(s) + \underline{\gamma}_k(s, s') + \underline{\beta}_{k+1}(s') \right) \\ & - \max_{\substack{s \to s' \\ C_{\kappa_c}=0}}{}^{*} \left(\underline{\alpha}_k(s) + \underline{\gamma}_k(s, s') + \underline{\beta}_{k+1}(s') \right) \end{aligned} \tag{4.37}$$

durchgeführt werden. In diesem Fall bezieht sich der max*-Operator auf alle Zustandsübergänge von s nach s', die an der entsprechenden Stelle κ_c mit dem Codebit 0 bzw. 1 verknüpft sind. Für die Berechnung von $L^o(b_k)$ folgt eine ähnliche Berechnungsvorschrift.

Der Max*-Log-MAP Decoder stellt die verbreitetste Form des MAP-Algorithmus zur SISO-Decodierung von Faltungscodes dar. Die Vorteile liegen in der Ersparnis der komplexen mathematischen Operationen sowie in der hohen Korrekturfähigkeit. Aus diesen Gründen wird der Max*-Log-MAP Decoder im weiteren Verlauf der Arbeit verwendet.

4.3.5 SCCC-Decoder

Der Vollständigkeit halber sind an dieser Stelle die nötigen Modifikationen des SISO-Decoders zur Decodierung von SCCCs erwähnt. Das in Abbildung 4.6 gezeigte Blockschaltbild verdeutlicht die Integration zweier MAP-Algorithmen zur Decodierung der verketteten Faltungscodes, die durch einen Interleaver π_r verknüpft sind. Die Decoder arbeiten nach dem in Abschnitt 4.3.3 beschriebenem Prinzip und unterscheiden sich lediglich durch den Faltungscode. Wie bereits in Abschnitt 3.2.2 erwähnt, handelt es sich beim inneren Faltungscode $\mathbf{C}_r$ um eine Rate-1 Code, der rein rekursiv ist. Während einer Turbo-Iteration tauschen der innere und äußere Algorithmus Zuverlässigkeitsinformation in Form von LLRs aus. Somit entsteht eine weitere Möglichkeit der Iteration, die im Weiteren als Decoder-Iterationen N_{it}^{D} bezeichnet wird.

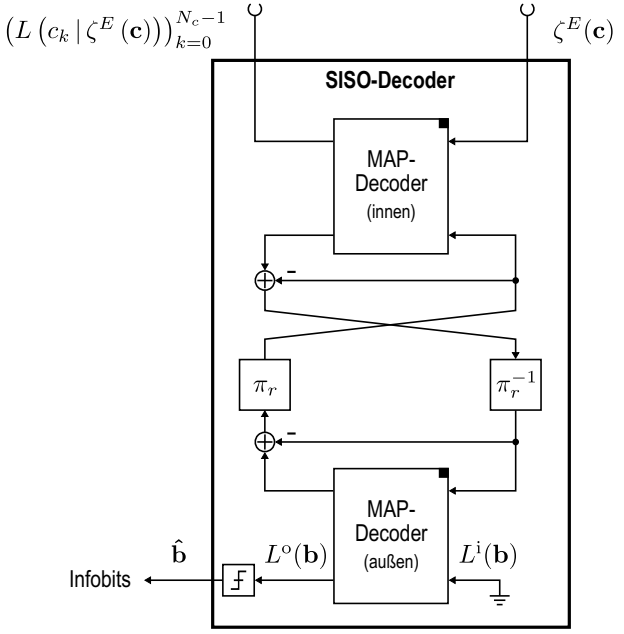


Abb. 4.6: Integration der MAP-Decoder zur Decodierung von SCCCs

4.4 Konvergenzanalyse

Aufgrund der iterativen Verarbeitung ist die Leistungsfähigkeit des Turbo-Entzerrers maßgeblich von dessen Konvergenzverhalten abhängig. Konvergiert das Verfahren, kann mit jeder Iteration die Bitfehlerrate reduziert werden. Die Konvergenz wird dabei sowohl von der Systemkonfiguration, wie z.B. dem Faltungscode, als auch vom SNR pro Bit beeinflusst. Die Analyse des Verhaltens stellt deshalb ein wichtiges Werkzeug zur Optimierung von Systemparametern dar.

Das 1999 von *ten* Brink [Bri99] für Turbo-Codes vorgeschlagene Extrinsic Information Transfer (EXIT) Diagramm ermöglicht es, die Konvergenz mithilfe der Informationstheorie grafisch darzustellen. Dazu wird die Transinformation (*engl.* Mutual Information) innerhalb des Turbo-Entzerrers an zwei Punkten berechnet und gegeneinander aufgetragen. Die Transinformation I^E wird zwischen den gesendeten Datenbits d_k und den LLRs $\lambda^E(d_k)$ nach der Entzerrung bestimmt und die Transinformation I^D zwischen den gesendeten Codebits c_k und den LLRs $\lambda^D(c_k)$ nach der Decodierung. Da die beiden Transinformationen durch die iterative Entzerrung und Decodierung voneinander abhängig sind, kann durch $I^E = f(I^D)$ das Konvergenzverhalten dargestellt werden.

Basierend auf dem Ansatz von *ten* Brink wird die Transinformation zwischen zwei Zufallsvariablen λ und x gemäß

$$I(\lambda, x) = \frac{1}{2} \sum_{x \in \{0,1\}} \int_{-\infty}^{\infty} p_\lambda(l \,|\, x) \cdot \mathrm{ld} \left(\frac{2 p_\lambda(l \,|\, x)}{p_\lambda(l \,|\, 0) + p_\lambda(l \,|\, 1)} \right) \mathrm{d}l \qquad (4.38)$$

bestimmt. Die Variable λ entspricht dem jeweiligen LLR am Ausgang der SISO-Komponenten. Ihre Verteilungsdichte p_λ wird für einen ausreichenden Stichprobenumfang durch ein Histogramm angenähert. Die Variable x entspricht dem gesendeten Daten- bzw. Codebit, dessen Zuverlässigkeit durch das LLR λ repräsentiert wird.

Die Berechnung der Transinformationen I^E und I^D findet zunächst getrennt voneinander statt. Die LLR-Eingänge der SISO-Komponenten werden dazu durch *Test*-LLRs gespeist. Diese werden so generiert, dass die Transinformation I^T zwischen den LLRs und den Daten- bzw.

Codebits von 0 bis 1 variiert wird. Aus den berechneten LLRs am Ausgang wird die Transinformation I^E bzw. I^D bestimmt und der Konvergenzverlauf $I^E = f^E(I^T)$ bzw. $I^D = f^D(I^T)$ aufgezeichnet.

Nach der getrennten Analyse des Konvergenzverhaltens können die berechneten Transinformationen und Konvergenzverläufe zur Evaluation des Turbo-Entzerrers herangezogen werden. Die Transinformation I^D am Ausgang des Decoders wird dazu auf der x-Achse und die des Entzerrers I^E auf der y-Achse aufgetragen. Die Darstellung erfolgt mithilfe der bereits bestimmten Verläufe gemäß

$$\begin{aligned} I^E &= f^E(I^D) \\ I^D &= f^D(I^E) \ . \end{aligned} \tag{4.39}$$

Der Verlauf der Konvergenz des Turbo-Entzerrers $I^E = f(I^D)$ lässt sich schließlich durch das vertikale und horizontale Traversieren der zwei kontinuierlichen Konvergenzverläufe konstruieren. Abbildung 4.7 zeigt exemplarisch das EXIT-Diagramm für eine Turbo-Entzerrung.

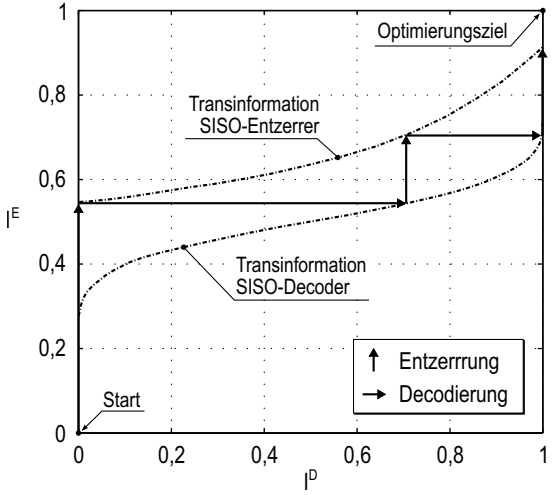


Abb. 4.7: EXIT-Diagramm eines Turbo-Entzerrers

Beginnend im Punkt (0,0) wird die Entzerrung durch eine Vertikale und die Decodierung durch eine Horizontale abgebildet. Zur Optimie-

rung des Turbo-Entzerrers sind die Verläufe von SISO-Entzerrer und -Decoder so zu entwerfen, dass das Optimierungsziel im Punkt (1,1) mit möglichst wenigen Iterationen erreicht wird.

4.5 Leistungsfähigkeit

Die Leistungsfähigkeit des SC-MMSE-FD Turbo-Entzerrers wird im Folgenden für die in Tabelle 4.1 aufgelisteten Kanalmodelle demonstriert und der Einfluss von Systemparametern anhand von EXIT-Diagrammen und Bitfehlerratenkurven gezeigt. Die Länge einer Infobit-Sequenz ist festgelegt auf $N_b = 1024$. Für das Interleaving wird ein einfacher Zufallsinterleaver verwendet. Die Simulationsergebnisse wurden über 10^8 Bits gemittelt.

Tab. 4.1: Kanalmodelle

Kennwert	`h_nofading`	`h_nlos50` (Non-Line of Sight)	`h_los130` (Line of Sight)
Systembandbreite	10 MHz		
Power-Delay Profile	Exponential Decay, $\tau_{\mathrm{DS}} = 0{,}5\,\mu\mathrm{s}$		
Tapped-Delay-Line Parameter	24 Koeffizienten, Verzögerungen: 0 µs, 0,1 µs, …, 2,3 µs		
Fading-Modell	Kein Fading	Rayleigh	Rice♠
Max. Rel'geschw.	/	50 km/h	130 km/h
Max. Dopplerversch.	/	18,52 Hz	48,15 Hz

♠: Leistungsverhältnis zwischen direktem Pfad und Umwegen: 8; Dopplerverschiebung des direkten Pfads: 3,7 Hz.
Alle Kanäle wurden auf eine mittlere Dämpfung von 0 dB normiert.

Turbo-Iteration Aus den Abbildungen 4.8 und 4.9 wird ersichtlich, dass sich die Bitfehlerwahrscheinlichkeit der decodierten Infobit-Folge $\hat{b}_k$ am Ausgang des SC-MMSE-FD Turbo-Entzerrers mit steigender Anzahl an Turbo-Iterationen reduziert. Für das Kanalmodell

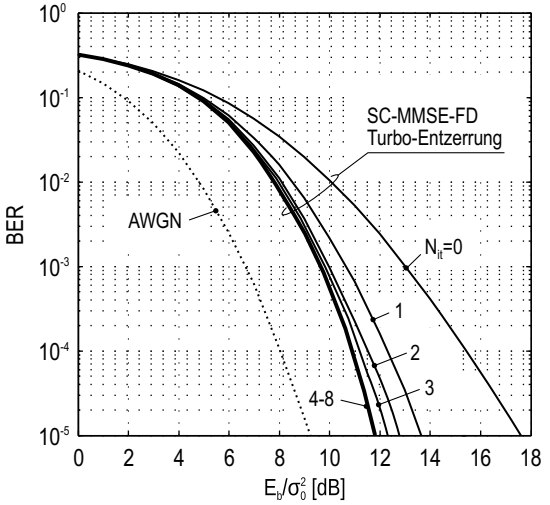


Abb. 4.8: Bitfehlerkurven für `h_los130` Kanalmodell, 16-QAM, 8 Turbo-Iterationen, SCC mit Faltungscode $\mathbf{C} = [5, 7]_8$. *Im Vergleich*: AWGN-Kanal ohne Fading.

`h_nofading` ohne Fading (s. Abb. 4.9) kann der Einfluss der Eigeninterferenz sogar vollständig durch einen SCC-Encoder eliminiert werden. Darüber hinaus ermöglicht die serielle Verkettung von Faltungcodes durch einen SCCC-Encoder (s. Abs. 3.2.2) eine Verbesserung, die aufgrund der *zusätzlichen* rekursiven Code-Komponente $\mathbf{C}_r$ mit Rate-1 zur Überschreitung der Leistungsfähigkeit im Falle eines AWGN-Kanals führen kann (s. Abb. 4.9) [Gro11].

Encoder Die Leistungsfähigkeit eines SC-MMSE-FD Turbo-Entzerrers wird maßgeblich durch die Korrekturfähigkeit des Fehlerschutzes bestimmt. Der Faltungscode beeinflusst das Konvergenzverhalten und ermöglicht somit individuelle, applikationsspezifische Systementwürfe. Das EXIT-Diagramm in Abbildung 4.10 verdeutlicht die Wirksamkeit von seriell verketteten Faltungscodes. Während für SCCs die Transinformation am Ausgang des Decoders nahezu gleichförmig mit I^T ansteigt, weisen SCCCs sprungartige Verläufe auf. Dies ist auf die

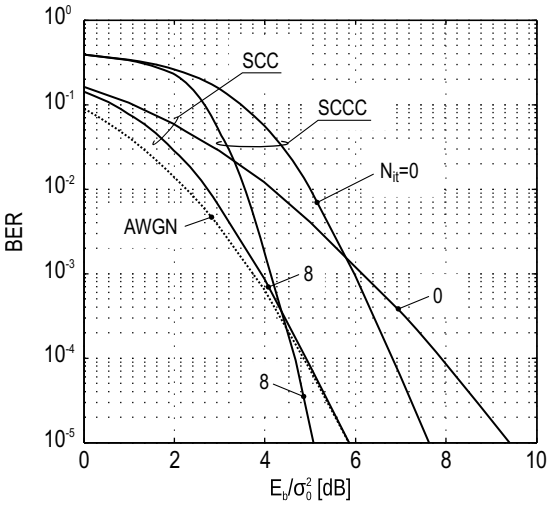


Abb. 4.9: Bitfehlerkurven für Kanalmodell `h_nofading` ohne Fading, BPSK, 8 Turbo-Iterationen, SCC mit Faltungscode $\mathbf{C} = [5,7]_8$; SCCC mit Faltungscode $\mathbf{C} = [5,7]_8$ bzw. $\mathbf{C}_r = [2;3]_8$ (innerer Code), eine Decoder-Iteration. *Im Vergleich*: AWGN-Kanal ohne Fading für SCC.

Fehlerfortpflanzung des rekursiven Rate-1 Codes zurückzuführen, der erst ab einem gewissen SNR pro Bit zur Korrekturfähigkeit des SCCCs beiträgt. Wie im Falle der Turbo-Iterationen N_{it} steigt die Transinformation am Ausgang des Decoders mit jeder Decoder-Iteration N_{it}^D und somit die Zuverlässigkeit der decodierten Infobit-Sequenz.

Modulation Der Übergang zu spektral effizienteren, höherstufigen Modulationsverfahren führt zu einer Reduzierung des SNR pro Bit am Empfänger. Dieser grundsätzliche Zusammenhang führt zu einer Verschiebung der Bitfehlerratenkurven in Abbildung 4.8 bzw. der Konvergenzverläufe in Abbildung 4.11. Der prinzipielle Verlauf der Konvergenz wird marginal durch das Gray-Mapping und das nichtlineare Soft-Demapping/Mapping beeinflusst.

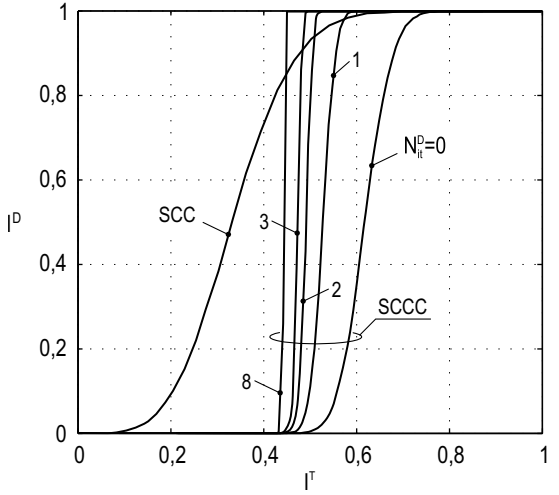


Abb. 4.10: EXIT-Diagramm eines SCCs und SCCCs für verschiedene Decoder-Iterationen und Coderate $r = \frac{1}{3}$. Faltungscode(s): $\mathbf{C} = [13, 15, 17]_8$ bzw. $\mathbf{C}_r = [2; 3]_8$ (innerer Code).

Kanalmodell Der Einfluss der Kanalmodelle ist in Abbildung 4.12 hervorgehoben. Für eine geringe Anzahl an Iterationen und Line-of-Sight ist tendenziell eine geringere Bitfehlerwahrscheinlichkeit zu erwachten. Mit steigender Anzahl an Turbo-Iterationen reduziert sich jedoch dieser Vorteil und führt zu einem ähnlichen Leistungsvermögen. Wie bereits erwähnt konvergiert im Falle eines frequenzselektiven Kanals ohne Fading-Effekte die Bitfehlerrate gegen die eines AWGN-Kanals.

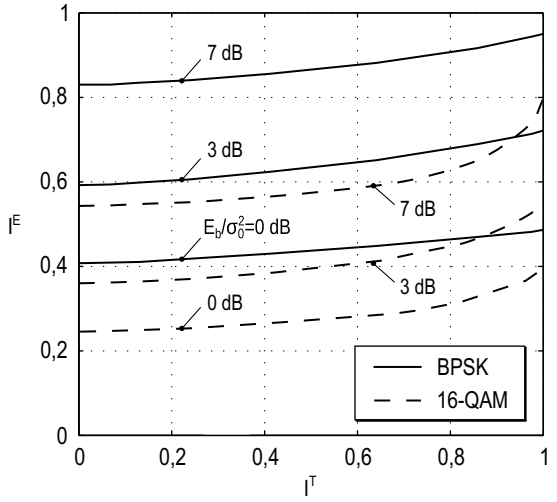


Abb. 4.11: EXIT-Diagramm des SC-MMSE-FD Entzerrers für verschiedene SNR pro Bit E_b/σ_0^2. Aufgrund der Normierung ist der Verlauf für alle Fading-Kanäle identisch.

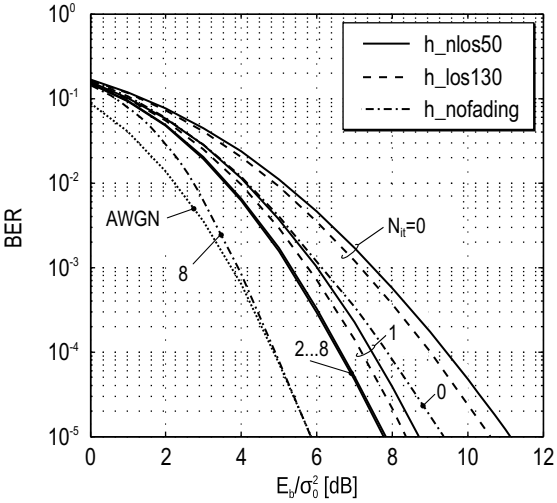


Abb. 4.12: Bitfehlerkurven für `h_nlos50` und `h_los130` Kanalmodell, BPSK, 8 Turbo-Iterationen, SCC mit Faltungscode $\mathbf{C} = [5, 7]_8$.

KAPITEL 5

Festkommaaspekte

Die Analyse von Festkommaaspekten ist ein wesentlicher Schritt zur Implementierung der SC-MMSE-FD Turbo-Entzerrung in Festkommaarithmetik. Die Notwendigkeit dieser Analyse wurde bereits in Kapitel 2 erläutert und spielt besonders für Logiken wie FPGAs eine wichtige Rolle. Der durch die Festkommaarithmetik *limitierte Dynamikbereich* der Zahlendarstellung verursacht Fehler, die sich auf die Leistungsfähigkeit der Turbo-Entzerrung auswirken. Neben dem Konvergenzverhalten der iterativen Entzerrung und Decodierung wird als Konsequenz auch die erzielbare Bitfehlerrate beeinträchtigt.

Artikel über Festkommaimplementierungen von SISO-Decodern existieren bereits in der Literatur und wurden vorwiegend im Kontext der Turbo-Codes veröffentlicht. Die Ergebnisse dieser Beiträge werden in dieser Arbeit diskutiert und zur Umsetzung des Decoders herangezogen. An den entsprechenden Stellen wird auf die Literatur verwiesen. Ebenso liegen bereits Veröffentlichungen zur Implementierung von Turbo-Entzerrern auf Festkomma-DSPs [BLL04] oder ASICs [SFS11] vor. Diese Untersuchungen beschränken sich jedoch auf das Ausnutzen der gesamten Wortbreite oder das simulative Bestimmen der Festkommadarstellungen für eine spezifische Systemkonfiguration. Die in dieser Arbeit untersuchten Festkommaaspekte heben sich von diesen Veröffentlichungen ab, da sie zur Dimensionierung von Darstellungen für beliebige Konfigurationen dienen.

Das Kapitel gliedert sich wie folgt: Nach einer Einführung in die Grundlagen der Festkommaarithmetik in Abschnitt 5.1 wird in Abschnitt 5.2 der Einfluss einer Festkommadarstellung auf die im Turbo-Entzerrer ausgetauschten LLRs untersucht. Die einzelnen SISO-Komponenten werden dazu zunächst als *perfekt* modelliert[1]. Auf Basis der Ergebnisse folgt die detaillierte Analyse der einzelnen Komponenten. In Abschnitt 5.3 werden die Fehler durch eine Festkommaarithmetik für den SC-MMSE-FD Entzerrer untersucht. Effiziente Festkommadarstellungen werden für verschiedene Modulationsverfahren abgeleitet und die Ergebnisse anhand einer FPGA-Implementierung verifiziert. Abschnitt 5.4 widmet sich dem Max*-Log-MAP Decoder und untersucht die Fehler in Abhängigkeit vom Faltungscode und von der Encoder-Struktur. Eine GPP-Implementierung verdeutlicht den Nutzen der Analysen auch für Prozessoren mit Gleitkommaarithmetik. Abschließend sind in Abschnitt 5.6 alle Ergebnisse zur Festkommaimplementierung des SC-MMSE-FD Turbo-Entzerrers zusammengefasst.

5.1 Grundlagen

Durch die Festkommaarithmetik werden zwei Arten von Fehlern verursacht. Ein *Quantisierungsfehler* entsteht aufgrund der endlichen Wortbreite[2] und der damit verbundenen endlichen Anzahl an Amplitudenstufen. Dies führt dazu, dass nicht jeder Zahlenwert durch eine Festkommazahl repräsentiert werden kann. Die Quantisierung auf die nächste Amplitudenstufe äußert sich als zusätzliches Rauschen im Turbo-Entzerrer. Die in dieser Arbeit betrachtete Festkommaarithmetik rundet Zahlenwerte auf die nächstgelegene Amplitudenstufe (*engl.* Rounding), da diese Rundungsmethode die geringste Rauschvarianz erzielt [RG75].

Der zweite Fehler entsteht aufgrund der Amplitudenbegrenzung (*engl.* Clipping) durch die Festkommaarithmetik (s. Tab. 2.3). Das heißt,

[1] Die internen Simulationen werden mit 64-bit-Gleitkommagenauigkeit durchgeführt.

[2] Anzahl an Bits pro Zahlenwert

dass Zahlenwerte auf den Maximalwert der Festkommadarstellung begrenzt werden. Dieser *Sättigungsfehler* führt zu nichtlinearen Verzerrungen im Turbo-Entzerrer.

Da die SC-MMSE-FD Entzerrung und die Max*-Log-MAP Decodierung einen unabhängigen Beitrag zur Konvergenz des Turbo-Entzerrers leisten (s. Abs. 4.4), kann die Analyse der Festkommaaspekte ebenfalls separat geschehen. Die Auswirkungen des Quantisierungs- und Sättigungsfehlers werden dazu für jede Komponente einzeln untersucht und quantitative Abhängigkeiten hervorgehoben. Mithilfe dieser Ergebnisse werden *effektive* Festkommadarstellungen abgeleitet, die einen Kompromiss zwischen Leistungsfähigkeit und Wortbreite ermöglichen. *Bemerkung:* Um eine isolierte Analyse des SC-MMSE-FD Entzerrers zu ermöglichen, wurde angenommen, dass senderseitig ein Rate-1 Code verwendet wird.

Notation und Vorbemerkung Die in dieser Arbeit verwendete Notation zur Festkommadarstellung wurde bereits in Abschnitt 2.1.2 eingeführt und erläutert. Die Wortbreite N setzt sich aus n Nachkommabits und m Integerbits zusammen. Für eine vorzeichenbehaftete und vorzeichenlose Darstellung gilt

$$\begin{aligned} \texttt{Fix\_N\_n}: &\quad N = m + n + 1, \quad \text{bzw.} \\ \texttt{uFix\_N\_n}: &\quad N = m + n \,. \end{aligned} \tag{5.1}$$

Eine wesentliche Eigenschaft der Darstellungen ist ihre Dynamik. Der Dynamikbereich wird durch das logarithmische Verhältnis von größter zu kleinster darstellbarer Festkommazahl berechnet und beträgt

$$D = 20 \log \left(2^N\right) \approx 6{,}02N \,. \tag{5.2}$$

Der Dynamikbereich ist somit einzig von der Wortbreite N abhängig. Aus theoretischer Sicht stellt demnach N den einzigen relevanten Parameter dar. Für eine Implementierung könnten z.B. alle Zahlenwerte auf Ganzzahlen skaliert werden, was die Notwendigkeit eines Festkommas, resp. Nachkommastellen, hinfällig machen würde. Das *virtuelle* Komma ermöglicht lediglich eine Skalierung, um die Festkommadarstellung an die Amplitude der Zahlenwerte anzupassen. Nichtsdestotrotz wird

zur Analyse der Festkommaaspekte das Festkomma verwendet, da es aus Gründen der Anschaulichkeit sinnvoller erscheint. Die Ergebnisse der Untersuchungen werden davon *nicht* beeinflusst.

Ein wichtige Vorüberlegung für die hier durchgeführte Analyse ist, dass lediglich Symbole bzw. Variablen betrachtet werden, die einen signifikanten Ressourcenverbrauch, im Sinne von Speicher oder Logikfläche, im Turbo-Entzerrer verursachen. Als Beispiel diene eine LLR-Sequenz der Länge 1024, die vom SC-MMSE-FD Entzerrer zum Max*-Log-MAP Decoder weitergegeben wird. Es macht einen großen Unterschied, ob für die Festkommaimplementierung eine generische Darstellung gewählt wird, z.B. `Fix_16_8`, oder, wie sich zeigen wird, eine Festkommadarstellung durch `Fix_4_0` vollkommen ausreicht. Die Speicherersparnis beträgt in diesem Fall mehr als 12 kbit. Parameter oder Konstanten, deren Beitrag zum Gesamtverbrauch vernachlässigbar ist, werden für die Untersuchungen durch generische Darstellung repräsentiert.

5.2 Log-Likelihood Ratio

Ein LLR entspricht der Entscheidungszuverlässigkeit bezüglich eines Daten- bzw. Codebits und wird zum Austausch der extrinsischen Information zwischen Decoder und Entzerrer verwendet. Die Darstellung wurde bereits in Abschnitt 3.3.2 zur Herleitung des Optimalempfängers eingeführt und lautet allgemein

$$L(b) = \ln\left(\frac{P(b=1)}{P(b=0)}\right) \quad \in \mathbb{R}\,. \tag{5.3}$$

Die Wahrscheinlichkeit, ob das Bit b eine 1 oder 0 ist, wird ins Verhältnis gesetzt und logarithmiert. Mithilfe von $P(b=0) = 1 - P(b=1)$ lässt sich der Verlauf von (5.3) darstellen und ist in Abbildung 5.1 gezeigt. Im Bereich der maximalen Unsicherheit ($P(b=1) \approx P(b=0)$) weist das LLR Werte kleiner 1 auf während es für hohe Wahrscheinlichkeiten eines Ereignisses sehr große Werte annimmt. Die nötige Festkommadarstellung für die LLRs $\zeta^D(\mathbf{d})$ am Eingang des Entzerrers sowie für die LLRs $\zeta^E(\mathbf{c})$ am Eingang des Decoders wird im Folgenden analysiert.

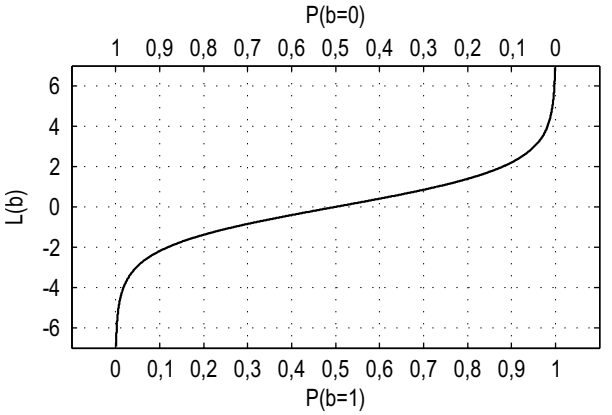


Abb. 5.1: Verlauf des LLRs

5.2.1 Entzerrereingang

Die Festkommadarstellung der LLRs $\zeta^D(\mathbf{d})$ führt zu Quantisierungs- und Sättigungsfehlern, die sich auf die gesamte Leistungsfähigkeit der SC-MMSE-FD Entzerrung auswirken. Zur Analyse der Fehlereinflüsse wird das EXIT-Diagramm herangezogen und der Verlauf für unterschiedliche Festkommadarstellungen bewertet. Das Soft-Mapping, die eigentliche Entzerrung sowie das Soft-Demapping werden dazu als perfekt modelliert.

Die exemplarischen Ergebnisse in Abbildung 5.2 zeigen, dass bereits 3 Integerbits ausreichen und keine Nachkommabits nötig sind, um einen nahezu idealen Verlauf der Transinformation I^E zwischen den Datenbits d_i und den LLRs $\lambda^E(d_k)$ am Ausgang des Soft-Demappings zu erhalten. Die vorzeichenbehaftete Darstellung der LLRs $\zeta^D(d_k)$ wird deshalb als `Fix_4_0` festgelegt. Die Darstellung wurde für weitere SNR pro Bit und Modulationsverfahren verifiziert und deckt sich mit den Ergebnissen in der Literatur [NFM09].

Die fehlende Notwendigkeit für Nachkommabits kann folgendermaßen erklärt werden: Existiert keine *a priori* Information von Seiten des Decoders, z.B. in der nullten Iteration, werden die Werte $\zeta^D(d_k) = 0$

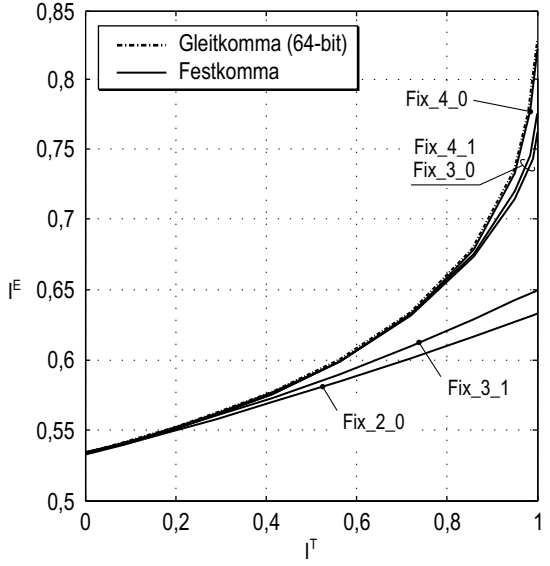


Abb. 5.2: EXIT-Diagramm des SC-MMSE-FD Entzerrers für verschiedene Festkommadarstellung der LLRs $\zeta^D(d_k)$ am Eingang (16-QAM, E_b/σ_0^2 = 4 dB).

korrekt durch den Festkommawert 0 repräsentiert. Steigt die Zuverlässigkeit der Information und dadurch auch die Amplitude der LLRs, werden kleine Werte im Bereich der maximalen Unsicherheit erneut auf Null quantisiert. Diese Quantisierung beeinflusst jedoch den Verlauf des SC-MMSE-FD Entzerrers nur marginal, da die Amplitude der LLRs im Falle der Konvergenz in der ersten Iteration bereits rasch ansteigt. Zur Verdeutlichung: Für eine Transinformation von I^T = 0,05 zwischen den Datenbits d_k und $\zeta^D(d_k)$ sind bereits ungefähr 35% der LLRs größer als 0,5 bzw. kleiner als −0,5 und werden deshalb auf die nächsten Stufen quantisiert. Für I^T = 0,5 sind es schon 90%.

Dass die Konvergenz für hohe Zuverlässigkeiten nur noch von der Anzahl an Integerbits bestimmt wird, lässt sich durch den exponentiellen Anstieg des LLRs in diesem Bereich erklären (s. Abb. 5.1). Mit m = 3 ist die Amplitude durch 7 bzw. -8 begrenzt (s. Tab. 2.3). Dies entspricht bereits $P(b = 1) = 0{,}9991$ bzw. $P(b = 0) = 0{,}9997$.

5.2.2 Decodereingang

Der Einfluss einer Festkommaarithmetik auf die Leistungsfähigkeit des Decoders wird mithilfe der Bitfehlerrate der decodierten Infobits $\hat{b}_k$ und des EXIT-Diagramms ermittelt. Der Decoder wird isoliert vom Turbo-Entzerrer analysiert und lediglich die LLRs $\zeta^E(c_k)$ am Eingang werden durch Festkommazahlen dargestellt. Die Decodierung selbst wird zunächst als *perfekt* modelliert. Die Untersuchungen sind gegenüber denen im vorherigen Abschnitt komplexer und abhängig vom Faltungscode und der Encoder-Struktur (SCC, SCCC).

Als Qualitätsanforderung wird festgelegt, dass durch die Festkommadarstellung eine Verschlechterung der Bitfehlerratenkurven von maximal 0,1 dB E_b/σ_0^2 im relevanten Bitfehlerratenbereich zwischen 0,5 bis 10^{-5} zulässig ist. Für den Verlauf der Transinformation zwischen den LLRs $\lambda^D(c_k)$ und den Codebits c_k gilt, dass der Fehler gegenüber einer Gleitkommadarstellung kleiner als 10^{-1} (L2-Norm) sein muss. Da beide Darstellungsformen der Leistungsfähigkeit korreliert sind, wurden stets beide Anforderungen für eine gegebene Systemkonfiguration erfüllt bzw. nicht erfüllt.

Abbildung 5.3 zeigt die Bitfehlerrate und das EXIT-Diagramm eines SCC mit rekursivem Faltungscode $\mathbf{C} = [13, 15, 17; 13]_8$ für drei verschiedene Festkommadarstellungen. Die Kurven wurden simulativ ermittelt. Die Analyse ergibt, dass bereits eine `Fix_5_1` Darstellung der LLRs $\zeta^E(c_k)$ die Qualitätsanforderungen erfüllt. Werden weniger Nachkommabits verwendet (z.B. `Fix_5_0`) resultiert eine Verschlechterung der Bitfehlerrate für geringe SNR pro Bit. In diesem Bereich weist das LLR besonders kleine Werte auf, die auf 0 quantisiert werden und keinen Beitrag zur Decodierung liefern. Werden zu wenige Integerbits (z.B. `Fix_4_1`) veranschlagt, führt dies zu einer Degradation der Bitfehlerrate für hohe SNR pro Bit. Die in diesem Fall begrenzte darstellbare Ereigniswahrscheinlichkeit von maximal $P(b = 1) = 0{,}9991$ bzw. $P(b = 0) = 0{,}9997$ wirkt sich dabei direkt auf die Bitfehlerwahrscheinlichkeit aus.

Die gleichen Analysen wurden für eine Reihe unterschiedlicher Faltungscodes und Decoder durchgeführt (s. Tab. 5.1). In allen untersuchten Fällen zeigte sich, dass eine `Fix_5_1` Darstellung der LLRs

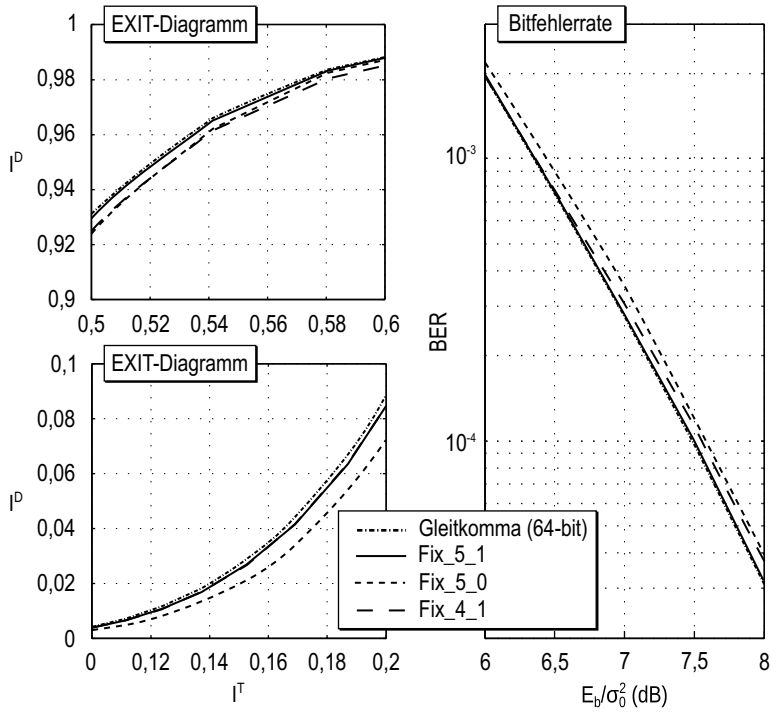


Abb. 5.3: EXIT-Diagramm und Bitfehlerraten eines SCCs mit Faltungscode $\mathbf{C} = [13, 15, 17; 13]_8$ für verschiedene Festkommadarstellung der LLRs $\zeta^E(c_k)$ am Eingang.

$\zeta^E(c_k)$ zur Erfüllung der aufgestellten Qualitätsanforderungen ausreicht. Das Ergebnis deckt sich mit denen in der Literatur [MB00]. Aus diesen Gründen wird deshalb im weiteren Verlauf der Analyse eine `Fix_5_1` Festkommadarstellung der LLRs $\zeta^E(c_k)$ verwendet.

5.3 SC-MMSE-FD Entzerrer

Der SC-MMSE-FD Entzerrer wurde bereits in Abschnitt 4.2 hergeleitet und das erforderliche Soft-Mapping bzw. Soft-Demapping näher

Tab. 5.1: Leistungsfähigkeit des Max*-Log-MAP Decoders für eine `Fix_5_1` Darstellung der LLRs $\zeta^E(c_k)$

Code	N_{it}^D	EXIT	BER
SCC $[5,7]_8$	/	0,008	0,03
SCC $[133,171]_8$	/	0,011	0,02
SCC $[13,15,17;13]_8$	/	0,012	0,02
SCCC $[5,7]_8$	0	0,011	0,03
	1	0,011	0,02
	3	0,015	0,02
	8	0,017	0,02
SCCC $[133,171]_8$	0	0,011	0,03
	1	0,012	0,02
	3	0,015	0,03
	8	0,014	0,02
SCCC $[13,15,17;13]_8$	0	0,014	0,03
	1	0,017	0,01
	3	0,018	0,02
	8	0,021	0,02

EXIT: Fehler im EXIT-Diagramm (L2-Norm)
BER: Max. Verlust (dB) an E_b/σ_0^2 im BER-Diagramm
Für SCCC: $\mathbf{C}_r = [2;3]_8$, π_r Zufallsinterleaver

erläutert. Eine Implementierung des Entzerrers in Festkommaarithmetik erfordert eine Festkommadarstellung für jedes verarbeitete Symbol. Aus diesem Grund gelten die in Abbildung 5.4 angegebenen Bezeichnungen für diesen Abschnitt.

Für die Analyse wird angenommen, dass die durch die Festkommaarithmetik bedingten Fehler bei der Durchführung der (inversen) FFT vernachlässigt werden können. Dies setzt voraus, dass eine Anpassung des Dynamikbereichs nach jeder Butterfly-Operation erfolgt. Diese Annahme wird für einen konfigurierbaren FFT IP-Core[1] in Abschnitt 5.3.3 näher untersucht.

[1] Intellectual Property (IP) Core - Wiederbenutzbarer Teil eines Chipdesigns

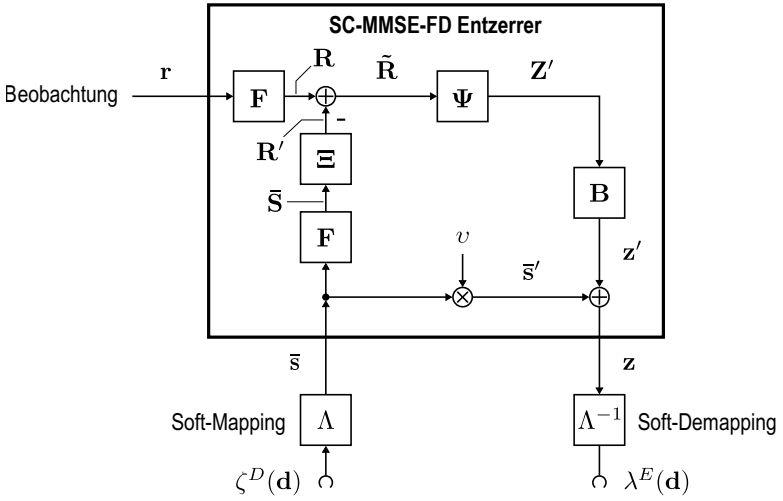


Abb. 5.4: Blockschaltbild des SC-MMSE-FD Entzerrers

5.3.1 Quantisierungsfehler

Zur Analyse des Quantisierungsfehlers wird zunächst ein analytisches Fehlermodell [RG75] aufgestellt. Eine reelle Zahl x wird aufgrund der endlichen Anzahl an Amplitudenstufen auf die Zahl

$$[x]_Q = x + q \tag{5.4}$$

quantisiert. Der Quantisierungsfehler q ist mittelwertfrei und gleichverteilt im Bereich Δ. Der Bereich entspricht der Höhe einer Amplitudenstufe und ist aufgrund des Roundings durch $\Delta = 2^{-n}$ gegeben (s. Tab. 2.3). Die Varianz des Fehlers bzw. die Fehlerleistung hängt somit *nur* von der Quantisierung ab und beträgt $\sigma^2 = \Delta^2/12$.

Das analytische Modell kann nun auf alle Symbole innerhalb des SC-MMSE-FD Entzerrers angewendet werden. Für die quantisierten Sym-

bole am Ausgang des Entzerrers folgt mit (4.10) und (5.4)

$$\begin{aligned}[\mathbf{z}]_Q =& \upsilon\bar{\mathbf{s}} + \mathbf{B}\boldsymbol{\Psi}\left(\mathbf{F}\mathbf{r} - \boldsymbol{\Xi}\mathbf{F}\bar{\mathbf{s}}\right) \\ &+ \left(\upsilon - \mathbf{B}\boldsymbol{\Psi}\boldsymbol{\Xi}\mathbf{F}\right)\mathbf{q}_{\bar{s}} \\ &- \mathbf{B}\boldsymbol{\Psi}\boldsymbol{\Xi}\mathbf{q}_{\bar{S}} \\ &+ \mathbf{B}\boldsymbol{\Psi}\mathbf{F}\mathbf{q}_r + \mathbf{B}\boldsymbol{\Psi}\mathbf{q}_R - \mathbf{B}\boldsymbol{\Psi}\mathbf{q}_{R'} + \mathbf{B}\boldsymbol{\Psi}\mathbf{q}_{\tilde{R}} \\ &+ \mathbf{B}\mathbf{q}_{Z'} + \mathbf{q}_{z'} + \mathbf{q}_{\bar{s}'} + \mathbf{q}_z \ .\end{aligned} \tag{5.5}$$

Der Vektor $\mathbf{q}_{(\cdot)}$ stellt jeweils das durch die Festkommadarstellung des Symbols $(\cdot)$ verursachte Quantisierungsrauschen mit der Varianz $\sigma^2_{(\cdot)}$ dar. Zu beachten ist hierbei, dass komplexwertige Zahlen verarbeitet werden. Die nötige Anpassung der Varianz auf $2\sigma^2_{(\cdot)}$ wurde in den Berechnungen berücksichtigt.

Die Umsetzung von 5.5 erfordert eine vollständige Realisierung des Kommunikationssystems aus Sender, Kanal und Empfänger. Zur vereinfachten Analyse des Quantisierungsfehlers kann an dieser Stelle erneut auf das äquivalente AWGN-Modell aus Abschnitt 4.2.3 zurückgegriffen werden. Das Modell ermöglicht es den Einfluss des Funkkanals und des Entzerrers auf die Modulationssymbole s_k nachzubilden und wird nun zur Modellierung der Quantisierungsfehler erweitert. Für das *quantisierte* AWGN-Modell folgt gemäß (4.12)

$$[z_k]_Q = \upsilon s_k + [w_k]_Q \ . \tag{5.6}$$

Da die Quantisierungsfehler mittelwertfrei sind, beeinflusst lediglich die Fehlerleistung die Varianz des Modells. Für diese folgt mit (5.5)

$$\begin{aligned}\mathrm{Var}\left\{[z_k]_Q\right\} =& \upsilon^2(\varphi - 1) + \upsilon \\ &+ \left(\upsilon^2 + \psi\right)\sigma^2_{\bar{s}} + \psi\sigma^2_{\bar{S}} \\ &+ \phi(\sigma^2_r + \sigma^2_R + \sigma^2_{R'} + \sigma^2_{\tilde{R}}) \\ &+ \sigma^2_{Z'} + \sigma^2_{z'} + \sigma^2_{\bar{s}'} + \sigma^2_z \ ,\end{aligned} \quad \left.\vphantom{\begin{aligned}a\\b\\c\\d\end{aligned}}\right\} \text{ Einfluss der Quantisierung} \tag{5.7}$$

mit $\psi = N_s^{-1}\mathrm{Spur}(\boldsymbol{\Psi}\boldsymbol{\Xi}\boldsymbol{\Xi}^H\boldsymbol{\Psi}^H)$ und $\phi = N_s^{-1}\mathrm{Spur}(\boldsymbol{\Psi}\boldsymbol{\Psi}^H)$. Die Varianz des Rauschens $[w_k]_Q$ setzt sich somit aus der Varianz des äquivalenten AWGN-Modells aus Abschnitt 4.2.3 und zusätzlichen Varianztermen, die durch die Quantisierung entstehen, zusammen.

Die Genauigkeit des Modells wird durch das EXIT-Diagramm in Abbildung 5.5 für unterschiedliche Modulationsverfahren und SNR pro Bit verdeutlicht. Die Anzahl an Nachkommastellen wurde in diesem Beispiel für alle Symbole gleich gewählt und Sättigungsfehler ausgeblendet.

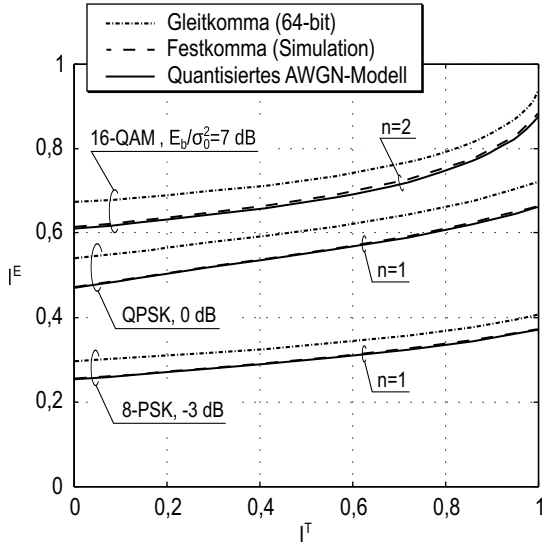


Abb. 5.5: EXIT-Diagramm des SC-MMSE-FD Entzerrers für unterschiedliche Implementierungen, Modulationsverfahren und Nachkommabits

Die Verläufe zeigen, dass der Quantisierungsfehler präzise durch das quantisierte AWGN-Modell nachgebildet werden kann. Während zur Berechnung der Kurven für die Gleitkomma- und Festkommaumsetzung das vollständige Kommunikationssystem simuliert werden musste, konnte der Verlauf des quantisierten AWGN-Modells analytisch in kurzer Zeit bestimmt werden. Aufgrund dieser Zeitersparnis und der exakten Nachbildung des Fehlers wird das Modell im Weiteren zur Dimensionierung der Festkommadarstellung für den SC-MMSE-FD Entzerrer verwendet.

Die Varianz des quantisierten AWGN-Modells kann darüber hinaus auch in Kansanens analytischer Methode zur Berechnung des EXIT-

Diagramms [KM07] verwendet werden. Für das SNR des äquivalenten Kanals gilt dann gemäß seiner Notation $\mathcal{L} = \frac{\upsilon}{\mathrm{Var}\{[z_k]_Q\}}$ ([KM07], Gl. 24).

5.3.2 Sättigungsfehler

Sättigungsfehler entstehen dann, wenn die Anzahl an Integerbits nicht ausreicht um die Amplitude einer reellen Zahl darzustellen. Die nichtlinearen Effekte, die sich auf den SC-MMSE-FD Entzerrer durch diese Amplitudenbegrenzung auswirken, werden im Folgenden untersucht. Da es sich bei den verarbeiteten Symbolen um kontinuierlich verteilte Zufallsvariablen handelt, ist es nötig, eine Wahrscheinlichkeit für den Fall der Sättigung anzugeben. Die weiteren Betrachtungen werden aufgrund der zirkukärsymmetrischen Verteilungen exemplarisch für den Realteil eines komplexwertigen Symbols durchgeführt. Die Ergebnisse sind auf den Imaginärteil übertragbar.

Die Wahrscheinlichkeit, dass die Amplitude einer reellen, kontinuierlichen Zufallsvariable x einen Sättigungswert Γ_{sat} überschreitet, ist durch

$$P_{\mathrm{sat}} = P(|x| > \Gamma_{\mathrm{sat}}) \tag{5.8}$$

gegeben. Zur Lösung dieses Zusammenhangs ist die volle Kenntnis der Verteilung von x nötig. Mithilfe der Tschebyscheff-Ungleichung [JW02] lässt sich die Berechnung des Sättigungswerts jedoch vereinfachen und durch das zweite Moment der Zufallsvariablen gemäß

$$\Gamma'_{\mathrm{sat}} < \frac{\mathrm{E}\{x^2\}}{P_{\mathrm{sat}}} \tag{5.9}$$

abschätzen. Da dieser Wert nur mit einer Wahrscheinlichkeit kleiner gleich P_{sat} überschritten wird, kann er als obere Schranke mit $\Gamma_{\mathrm{sat}} \leq \Gamma'_{\mathrm{sat}}$ für die Amplitudenbegrenzung angesehen werden. Obwohl andere Ungleichungen, wie z.B. die Chernoff-Ungleichung [Gut10], einen geringeren Schätzfehler zur Folge haben, bietet die Tschebyscheff-Ungleichung einen Kompromiss zwischen Genauigkeit und Berechnungskomplextität.

Mit der oberen Schranke Γ'_{sat} kann nun die Anzahl der nötigen Integerbits m berechnet werden. Hierbei wird diese erneut nach oben abgeschätzt, da der Maximalwert der Festkommadarstellung einer Zweierpotenz entspricht (s. Tab. 2.3). Für die Integerbits folgt somit

$$m = \lceil \mathrm{ld}(\Gamma'_{\mathrm{sat}}) \rceil \, . \tag{5.10}$$

Für die exakte Berechnung des Sättigungswerts und die Abschätzungen durch die Tschebyscheff-Ungleichung und Festkommadarstellung gilt demnach

$$\Gamma_{\mathrm{sat}} \leq \Gamma'_{\mathrm{sat}} \leq 2^m \, . \tag{5.11}$$

Zur Bestimmung des zweiten Moments in (5.9) wird angenommen, dass die Kanalimpulsantwort $\mathbf{h}$ auf $\mathrm{E}\{|h_l|^2\} = 1$ normiert ist und jeder Koeffizient der Verteilung $\mathcal{CN}(0,1)$ genügt. Mithilfe des Systemmodells (3.14) und den Ausführungen zum SC-MMSE-FD Turbo-Entzerrer in Abschnitt 4.2 können die Momente dann durch

$$\begin{aligned}
\mathrm{E}\left\{(\mathrm{Re}\{r_k\})^2\right\} &= 0{,}5\,\left(1+\sigma_0^2\right)\\
\mathrm{E}\left\{(\mathrm{Re}\{R_k\})^2\right\} &= 0{,}5\,N_s\left(1+\sigma_0^2\right)\\
\mathrm{E}\left\{(\mathrm{Re}\{\bar{S}_k\})^2\right\} &= 0{,}5\,N_s\varphi\\
\mathrm{E}\left\{(\mathrm{Re}\{R'_k\})^2\right\} &= 0{,}5\,N_s\varphi\\
\mathrm{E}\left\{(\mathrm{Re}\{\tilde{R}_k\})^2\right\} &= 0{,}5\,N_s\left(1+\sigma_0^2-\varphi\right)\\
\mathrm{E}\left\{(\mathrm{Re}\{Z'_k\})^2\right\} &= 0{,}5\,N_s\upsilon\\
\mathrm{E}\left\{(\mathrm{Re}\{z'_k\})^2\right\} &= 0{,}5\,\upsilon\\
\mathrm{E}\left\{(\mathrm{Re}\{\bar{s}'_k\})^2\right\} &= 0{,}5\,\upsilon^2\varphi\\
\mathrm{E}\left\{(\mathrm{Re}\{z_k\})^2\right\} &= 0{,}5\,\left(\upsilon^2\varphi+\upsilon\right)
\end{aligned} \tag{5.12}$$

approximiert werden. Diese Näherungen sind für die hier betrachteten Blocklängen ($N_s > 128$) gerechtfertigt, da die Summation innerhalb der DFT die Annahme des Zentralen Grenzwertsatzes [JW02] erlaubt. Die Zusammenhänge in (5.12) zeigen, dass die Momente vorwiegend von der Varianz des Rauschens σ_0^2, vom Parameter υ (s. Gl. (4.11)) und der mittleren Leistung φ der geschätzten Symbole $\bar{\mathbf{s}}$ am Eingang

des SC-MMSE-FD Entzerrers abhängen. Die DFT führt dabei zu einer Vergrößerung des zweiten Moments eines Symbols um den Faktor N_s. Dies hat direkte Auswirkungen für die Festkommadarstellung der Symbole im Frequenzbereich. Im Vergleich zum Zeitbereich werden bei der Darstellung im Frequenzbereich durch die Transformation

$$\Delta m = \left\lceil 1 - \frac{\mathrm{ld}(N_s)}{\mathrm{ld}(P_{\mathrm{sat}})} \right\rceil \tag{5.13}$$

Integerbits *mehr* benötigt (s. Tabelle 5.2).

Einen Spezialfall stellt die Anzahl an Integerbits für die Festkommadarstellung der geschätzten Symbole $\bar{s}_k$ dar. Das Soft-Mapping selbst begrenzt die Amplitude des Real- und Imaginärteils für die hier betrachteten Modulationsverfahren (BPSK, QPSK, 8-PSK, 16-QAM) auf 1. Die Anzahl an nötigen Bits entspricht somit maximal $m = 1$.

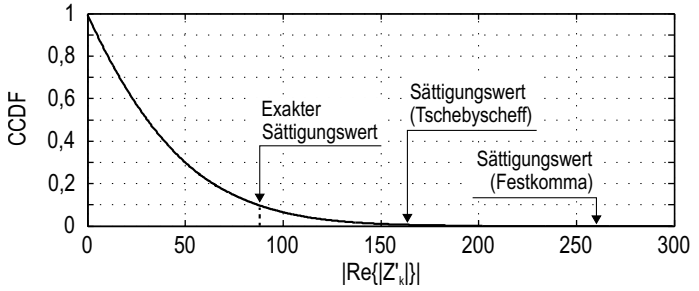


Abb. 5.6: Komplementäre Verteilungsfunktion für $|\mathrm{Re}\{Z'_k\}|$ mit verschiedenen Sättigungswerten für $P_{\mathrm{sat}} = 0{,}1$ ($N_s = 1024$, $E_b/\sigma_0^2 = 10\,\mathrm{dB}$, $\varphi = 0{,}1$)

Abbildung 5.6 zeigt exemplarisch die komplementäre Verteilungsfunktion (*engl.* Complementary Cumulative Distribution Function, CCDF) für $|\mathrm{Re}\{Z'_k\}|$. Der Verlauf wurde simulativ durch ein Histogramm ermittelt. Die Sättigungswerte für die verschiedenen Abschätzung sind eingezeichnet. Während die exakte Amplitudenbegrenzung für eine Sättigungswahrscheinlichkeit $P_{\mathrm{sat}} = 0{,}1$ bereits bei einem Wert von 80,9 auftritt, beträgt der Wert bei der Abschätzung durch die Tschebyscheff-Ungleichung 169,1. Zur Darstellung dieses Wertes sind wiederum 8 Integerbits nötig, was zu einem neuen Sättigungswert von 256

führt. Die diesem Wert entsprechende reale Sättigungswahrscheinlichkeit liegt im Umkehrschluss unter 0,01.

Die Tschebyscheff-Ungleichung ermöglicht es bei einer gegebenen Sättigungswahrscheinlichkeit den Sättigungswert und letztendlich die erforderliche Anzahl an Integerbits einfach abzuschätzen. Dieses Vorgehen wird deshalb zur Dimensionierung der Festkommadarstellung für eine gegebene Systemkonfiguration im folgenden Abschnitt verwendet.

5.3.3 FPGA-Implementierung

Zur Verifizierung der Ergebnisse aus den vorherigen Abschnitten und zur Untermauerung der Annahme, dass der FFT-Algorithmus intern keine Quantisierungs- oder Sättigungsfehler verursacht, wird für eine gegebene Systemkonfiguration eine FPGA-Implementierung des SC-MMSE-FD Entzerrers erstellt. Die Dimensionierung einer *effizienten* Festkommadarstellung erfolgt dabei mithilfe des quantisierten AWGN-Modells und der Abschätzung durch die Tschebyscheff-Ungleichung. Effizient bedeutet in diesem Zusammenhang, dass die minimale Anzahl an Integer- und Nachkommabits bestimmt wird, die einen guten Kompromiss zwischen Leistungsfähigkeit und Ressourcenverbrauch ermöglicht. Die Systemkonfiguration basiert auf einer 16-QAM und Sequenzen der Länge $N_s = 1024$. Die Evaluation der Leistungsfähigkeit geschieht für den E_b/σ_0^2-Bereich von $-2\,\mathrm{dB}$ bis $14\,\mathrm{dB}$.

Nachkommabits Zunächst wird die nötige Anzahl an Nachkommabits für die verarbeiteten Symbole ermittelt. Da aufgrund des Soft Interference Cancellations Symbole unterschiedlicher Herkunft (Beobachtung, Decoder) subtrahiert bzw. addiert werden (s. Abb. 5.4), gilt es die Erzeugung von künstlichem Quantisierungsrauschen zu vermeiden. Als Beispiele werde das 16-QAM Symbol $0{,}9487 + j0{,}3162$ einmal mit $n = 3$ Nachkommabits $(1 + j0{,}375)$ und einmal mit $n = 2$ Bits $(1 + j0{,}25)$ dargestellt. Die Subtraktion der unterschiedlich quantisierten Symbole liefert mit $j0{,}125$ einen Wert ungleich Null. Aus diesem Grund wird die gleiche Anzahl an Nachkommabits für alle Symbole im SC-MMSE-FD Entzerrer verwendet.

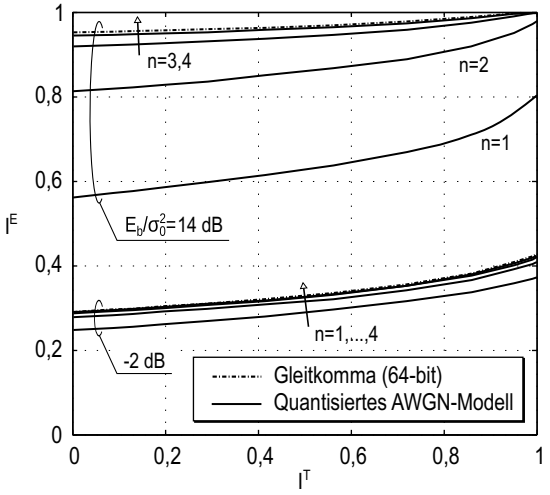


Abb. 5.7: EXIT-Diagramm des SC-MMSE-FD Entzerrers für verschiedene Nachkommabits n (16-QAM)

Für die Systemkonfiguration wird jeweils der Verlauf der Transinformation für verschiedene Werte n mithilfe des Modells aus Abschnitt 5.3.1 berechnet. Abbildung 5.7 zeigt die Ergebnisse für 1 bis 4 Nachkommabits. Es ist zu erkennen, dass $n = 4$ Nachkommabits ausreichen, um einen nahezu idealen Verlauf zu erhalten. Während für kleine SNR pro Bit der Einfluss der Nachkommabits eher gering ist, zeigt sich für große E_b/σ_0^2 eine deutliche Abhängigkeit der Transinformation. Zur Gewährleistung einer konstanten Leistungsfähigkeit wurde deshalb die Darstellung auf 4 Nachkommabits festgelegt. Für die Anzahl ist dabei die Blocklänge irrelevant. Lediglich die Stufigkeit des Modulationsverfahrens geht über die mittlere Leistung φ der geschätzten Symbole in die Modellierung mit ein.

Integerbits Zur Bestimmung einer effizienten Anzahl an Integerbits m wird zunächst die Sättigungswahrscheinlichkeit P_{sat} bestimmt. Abbildung 5.8 verdeutlicht, dass eine Wahrscheinlichkeit von $P_{\text{sat}} = 0{,}1$ die Transinformation marginal beeinflusst. Dieser Wert wurde für verschiedene Modulationsverfahren sowie Blocklängen verifiziert und

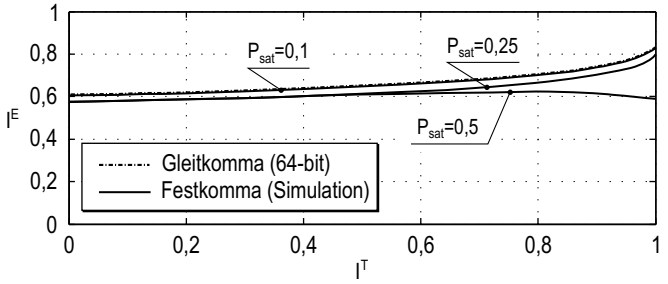


Abb. 5.8: EXIT-Diagramm des SC-MMSE-FD Entzerrers für verschiedene Sättigungswahrscheinlichkeiten P_{sat}. Quantisierungsfehler sind ausgeblendet.

kann als unabhängig davon angesehen werden. Mithilfe dieser Sättigungswahrscheinlichkeit, der Abschätzung (5.10) und den Näherungen (5.12) folgt die Berechnung von m für jedes Symbol im SC-MMSE-FD Entzerrer. Tabelle 5.2 fasst die Ergebnisse für die hier betrachteten Extremsituationen der Übertragung zusammen.

Tab. 5.2: Anzahl an Integerbits

Symbol	E_b/σ_0^2=−2 dB		E_b/σ_0^2=14 dB		**Max**
	φ klein	φ groß	φ klein	φ groß	
$\mathbf{r}$	2	2	2	2	**2**
$\mathbf{R}$	7	7	7	7	**7**
$\bar{\mathbf{S}}$	5	7	5	7	**7**
$\mathbf{R}'$	5	7	5	7	**7**
$\mathbf{R}''$	7	6	7	4	**7**
$\mathbf{Z}'$	6	7	7	10	**10**
$\mathbf{z}'$	1	2	2	5	**5**
$\bar{\mathbf{s}}'$	0	3	0	8	**8**
$\mathbf{z}$	1	3	2	8	**8**
klein: $0 < \varphi < 0{,}1$, groß: $\varphi > 0{,}95$					

Da die Festkommadarstellung für alle Rahmenbedingungen der Übertragung ausgelegt werden muss, wird jeweils das Maximum einer Zeile in Tabelle 5.2 als Wert festgelegt.

Tab. 5.3: Festkommadarstellungen

Symbol	n	m	N	D (dB)	Notation
$\mathbf{r}$	4	2	7	42,14	`Fix_7_4`
$\mathbf{R}$	4	7	12	72,25	`Fix_12_4`
$\bar{\mathbf{s}}$	4	1	6	36,12	`Fix_6_4`
$\bar{\mathbf{S}}$	4	7	12	72,25	`Fix_12_4`
$\mathbf{R}'$	4	7	12	72,25	`Fix_12_4`
$\mathbf{R}''$	4	7	12	72,25	`Fix_12_4`
$\mathbf{Z}'$	4	10	15	90,31	`Fix_15_4`
$\mathbf{z}'$	4	5	10	60,21	`Fix_10_4`
$\bar{\mathbf{s}}'$	4	8	13	78,27	`Fix_13_4`
$\mathbf{z}$	4	8	13	78,27	`Fix_13_4`

Umsetzung Die FPGA-Implementierung des SC-MMSE-FD Entzerrers wird mithilfe der *ISE Design Suite* von Xilinx [XIS14] für die in Tabelle 5.3 ermittelten Festkommadarstellungen erstellt. Die integrierte FPGA-Entwicklungsumgebung ermöglicht es, eine Implementierung zu entwerfen, zu analysieren und für verschiedene Xilinx FPGAs zu synthetisieren. Zur Modell-basierten Entwicklung bietet die Software darüber hinaus mit dem *System Generator for DSP* [XSG14] eine Erweiterung für MATLAB Simulink [SIM14] zum grafischen Entwurf komplexer Systeme. Diese wird für die Umsetzung des SC-MMSE-FD Entzerrers herangezogen und im Folgenden näher erläutert. Abbildung 5.9 zeigt die Top-Level Ansicht und die Beschaltung des SC-MMSE-FD Entzerrers in Simulink.

Mit dem Produkt LogiCORE IP Fast Fourier Transform v7.1 [XIP14] bietet Xilinx einen vorgefertigten IP-Core, der die Berechnung der (I)FFT zur Frequenzbereichsentzerrung übernimmt. Neben anwendungsspezifischen Parametern (z.B. FFT-Länge) lassen sich auch Details der Implementierung einstellen sowie die Architektur (Burst, Pipeline) bestimmen. Tabelle 5.4 zeigt die für diese Arbeit festgelegten Parameter und Abbildung 5.10 die Beschaltung des Simulink-Blocks. (I)FFT IP-Cores mit ähnlichem Funktionsumfang finden sich auch bei anderen FPGA-Herstellern [AIP14] oder bei OpenCores [OCo14]. Die übrige Signalverarbeitung des SC-MMSE-FD Entzerrers kann mithil-

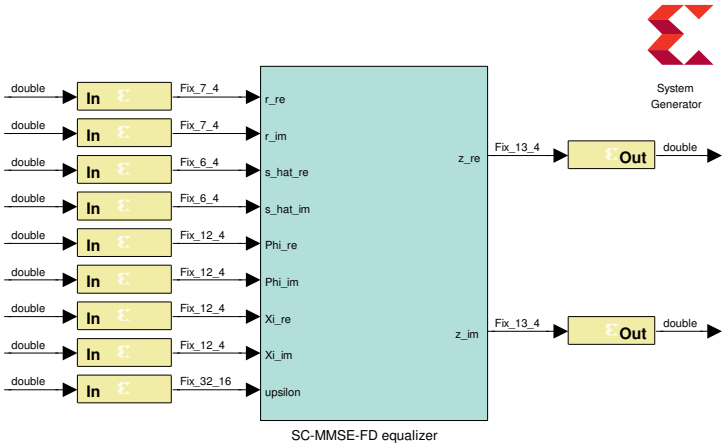


Abb. 5.9: Top-Level Ansicht des SC-MMSE-FD Entzerrermodells in Simulink

fe von einfachen Verzögerungsgliedern, Multiplizierern, Addierern und Subtrahierern sowie Blöcken, die zur Manipulation der Festkommadarstellung dienen, entworfen werden.

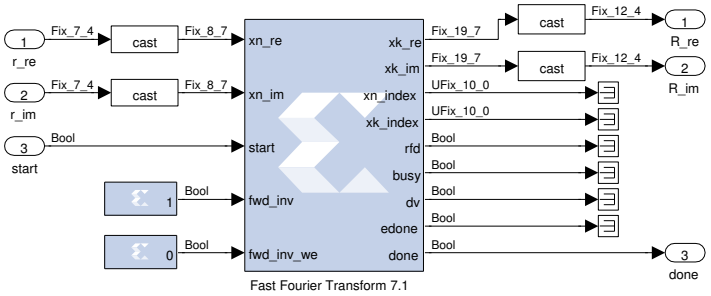


Abb. 5.10: Beschaltung des LogiCORE IP Fast Fourier Transform v7.1 Blocks in Simulink

Evaluation und Ergebnisse Die Implementierung des SC-MMSE-FD Entzerrers wird hinsichtlich zwei unterschiedlicher Gesichtspunkte evaluiert: Ressourcenverbrauch und Leistungsfähigkeit. Die Bemes-

Tab. 5.4: LogiCORE IP Fast Fourier Transform v7.1 Parametrisierung

Parameter	Wert
Länge	1024
Architektur	Pipelined, Streaming I/O
Phase Factor Width	16
Skalierung	Unscaled
Rounding	Convergent Rounding
Speicher	Block RAM, 1 Stage
Optimierung	3-muliplier structure, CLB logic

sung der benötigten FPGA-Ressourcen erfolgt durch die Anzahl der verwendeten Logikgatter. Im Falle von Xilinx FPGAs werden Logikgatter als *Slices* bezeichnet und in ihrer Funktionalität als Register (Flip-Flop) oder Lookup-Table (LUT) unterschieden. Des Weiteren werden durch spezielle Slices, sogenannte XtremeDSP48 Slices, unter anderem Multiplikationen ermöglicht. Zur Bestimmung des Ressourcenverbrauchs wird das Simulink-Modell des Entzerrers für die Xilinx FPGAs

- Kintex-7 XC7K410T [XK714] (Ettus Research USRP X310),
- Spartan-6 XC6SLX150 [XS614] (Ettus Research USRP B210) und
- Virtex-6 XC6VLX550T [XV614] (Nutaq µSDR420)

synthetisiert und das Protokoll des Place&Route-Vorgangs ausgewertet. Die FPGAs sind, wie oben angegeben, auf aktuellen SDRs der Hersteller Ettus Research [Ett14] und Nutaq [Nut14] integriert.

Die Ressourcenanalyse in Tabelle 5.5 zeigt, dass die ermittelten Festkommadarstellungen der Symbole im SC-MMSE-FD Entzerrer einen geringen Verbrauch ermöglichen. Konkret sind für die betrachtete Systemkonfiguration rund 25000 Flip-Flops, 26000 LUTs und ca. 60 Spezialgatter nötig. Dies entspricht einer Auslastung von weniger als 10% (Kintex-7, Virtex-6). Die Umsetzung des SC-MMSE-FD Entzerrers auf aktuelle FPGAs ist somit effizient möglich. Da die Anzahl an Integerbits lediglich logarithmisch mit der Blocklänge N_s steigt (s. Gl.

(5.10)), erhöht sich der Ressourcenverbrauch nur geringfügig für größere Sequenzlängen. Für eine Blocklänge von z.B. N_s = 4096 sind ca. 3 Bit pro Symbol zusätzlich nötig. Die in dieser Systemkonfiguration verwendete 16-QAM stellt für mobile Funkstandards bereits ein sehr hochstufiges Modulationsverfahren dar, was die Anzahl an Nachkommabits an dieser Stelle nach oben hin begrenzt. Für eine 64-QAM ist im Mittel pro Symbol mit einem Nachkommabit mehr zu rechnen.

Die Ergebnisse werden darüber hinaus mit einer *naiven*[1] Implementierung des Entzerrers verglichen. Für alle FPGAs konnte ein Anstieg des Ressourcenverbrauchs von mehr als 200% beobachtet werden. Im Falle des Spartan-6 wird der verfügbare Speicher sogar überschritten und der Place&Route-Vorgang abgebrochen.

Tab. 5.5: Ressourcenverbrauch

Anzahl	**Kintex-7**		**Spartan-6**		**Virtex-6**	
Slices	Naiv	**Analyse**	Naiv	**Analyse**	Naiv	**Analyse**
FFs	11%	24960 (4%)	31%	25132 (13%)	8%	24960 (3%)
LUTs	23%	26133 (10%)	66%♠	26080 (28%)	17%	26177 (7%)
DSP48s	7%	52 (3%)	81%	61 (33%)	13%	52 (6%)

Flip-Flop (FF); Lookup-table (LUT); Naiv: `Fix_32_16`

♠: Speicherüberlast (*engl.* Overmapped)

Die Leistungsfähigkeit der FPGA-Implementierung kann durch die Gegenüberstellung der Transinformation am Ein- und Ausgang mithilfe eines EXIT-Diagramms bestimmt werden. Da es Simulink ermöglicht die Funktionalität der Modellierung detailgetreu, bis auf das Timing, zu simulieren, wird das EXIT-Diagramm direkt in Simulink vor der eigentlichen Synthese ermittelt. *Bemerkung:* Im Protokoll des Place&Route Vorgangs wurden keine Timing-Probleme festgestellt. Aufgrund der langen Simulationszeit der FPGA-Modellierung wurden jedoch lediglich dedizierte Punkte im Diagramm bestimmt und mit den prognostizierten Ergebnissen der Analyse verglichen. Abbildung 5.11 zeigt die Übereinstimmung der Transinformationen und verifiziert damit die Genauigkeit der Modelle aus Abschnitt 5.3.1 und 5.3.2.

[1] Alle Symbole werden durch `Fix_32_16` repräsentiert.

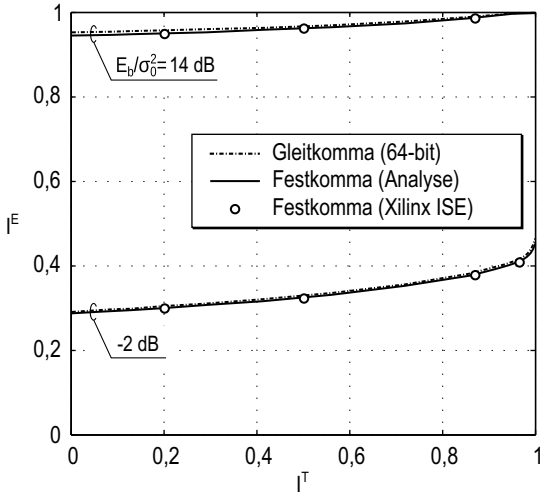


Abb. 5.11: EXIT-Diagramm des SC-MMSE-FD Entzerrers für verschiedene Implementierungen

Die Annahme, dass der FFT-Algorithmus keine zusätzlichen, durch die Festkommaarithmetik bedingten, Fehler verursacht, konnte ebenso bestätigt werden. Die Skalierung innerhalb des Fast Fourier Transform v7.1 IP-Cores wurde für die Implementierung *deaktiviert*, sodass sich nach jeder Butterfly-Operation die Anzahl an Integerbits um 1 Bit erhöhte. Dieser Mechanismus verhindert zwar einen Überlauf bei der Butterfly-Addition, hat aber auch einen höheren Ressourcenverbrauch zur Folge. Aus diesem Grund wurde die Implementierung auch für eine *aktivierte* Skalierung innerhalb der FFT-Algorithmen durchgeführt und die Leistungsfähigkeit sowie der Ressourcenverbrauch erneut evaluiert. Während dies eine geringfügige Ersparnis an Logikgattern ermöglichte, verschlechterte sich der Verlauf der Transinformation. Für den Kintex-7 betrug z.B. die Anzahl an Flip-Flops anstatt 24960 nur 23117. Da dies lediglich einer Gesamtersparnis von 1% entspricht, erscheint eine unskalierte Umsetzung der FFT-Algorithmen im Hinblick auf ein besseres Konvergenzverhalten sinnvoller.

5.4 Max*-Log-MAP Decoder

Wie in der Einleitung dieses Kapitels erwähnt, wurden SISO-Decoder bezüglich ihrer Festkommaaspekte bereits untersucht (z.B. [MPRZ99], [YW99], [MB00], oder [BGG03]). Die Veröffentlichungen entstanden überwiegend im Zusammenhang mit der Einführung des Universal Mobile Telecommunications Systems (UMTS), das als erster Funkstandard Turbo-Codes zur Fehlerkorrektur einsetzt. Obwohl die Untersuchungen für ASICs bzw. Very Large Scale Integration (VLSI) Architekturen durchgeführt wurden, können die Ergebnisse in angepasster Form auch für Logiken verwendet werden.

Die Decodierung der LLRs $\zeta^E(\mathbf{c})$ wurde in Abschnitt 4.3 für den Max*-Log-MAP Decoder beschrieben und die dafür erforderlichen Notationen sowie die Trellis-Darstellung eingeführt. Im Gegensatz zum SC-MMSE-FD Entzerrer mit zwei Eingängen verfügt der Decoder lediglich über einen Eingang. Für die LLRs $L^{\mathrm{i}}(c_k) = \zeta^E(c_k)$ wurde in Abschnitt 5.2 ermittelt, dass eine Darstellung durch `Fix_5_1` hinreichend ist. Ebenso konnte für die LLRs $\zeta^D(d_k)$ am Eingang des Entzerrers die Darstellung `Fix_4_0` bestimmt werden. Mithilfe dieser beiden Darstellungen ist es möglich, den Decoder für eine Festkommaarithmetik zu entwerfen. Die relevanten Vorgänge, die es hinsichtlich einer Festkommadarstellung zu analysieren gilt, sind die Berechnung der Metrik, der Zustandswahrscheinlichkeiten sowie der Soft- und Hard-Decision.

5.4.1 Metrik

Die logarithmierte Übergangswahrscheinlichkeit $\underline{\gamma}_k(s, s')$ von Zustand s nach s' wird mithilfe der LLRs $L^{\mathrm{i}}(c_k)$ gemäß

$$\underline{\gamma}_k(s, s') = \sum_{\kappa_c=0}^{n_c-1} \underbrace{\ln\left(\frac{1}{1 + e^{(1-2C_{\kappa_c})L^{\mathrm{i}}(c_{n_c k+\kappa_c})}}\right)}_{=\underline{P}_{\kappa_c}} \tag{5.14}$$

berechnet. *Bemerkung:* Da der Vorfaktor $\frac{1}{n_b}$, wie in (4.29) angegeben, das Ergebnis der Soft- und Hard-Decision nicht beeinflusst, wird er bereits bei der Implementierung vernachlässigt. Bei genauerer Betrachtung des Nenners wird ersichtlich, dass die Multiplikation im Exponenten mit $1-2C_{\kappa_c}$, je nach Transition, lediglich einen Vorzeichenwechsel der LLRs $L^{\mathrm{i}}(c_{n_c k+\kappa_c})$ bewirkt. Zusammen mit der Festkommadarstellung durch `Fix_5_1` folgt, dass die logarithmierte Wahrscheinlichkeit $\underline{P}_{\kappa_c}$ ebenfalls nur $2^5 = 32$ verschiedene Werte annehmen kann. Da zudem das Ergebnis stets kleiner gleich Null ist, kann eine vorzeichenlose Darstellung verwendet und das Minus in den Algorithmus verschoben werden. Bei der Implementierung werden die Werte in einer Lookup-Table abgelegt. Die nötige Festkommadarstellung dieser Werte wird im Weiteren untersucht.

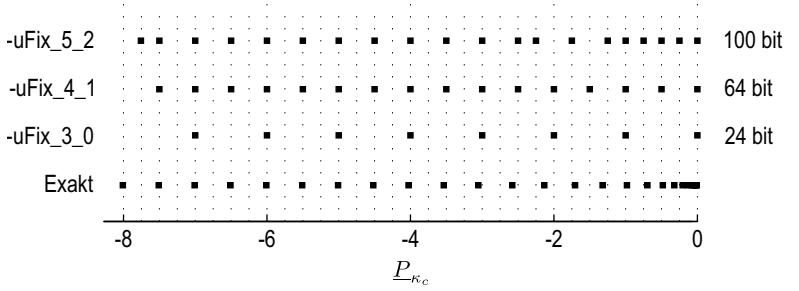


Abb. 5.12: Logarithmierte Wahrscheinlichkeiten $\underline{P}_{\kappa_c}$ in verschiedenen Festkommadarstellung für $L^{\mathrm{i}}(c_k)$ in `Fix_5_1` Darstellung. *Rechts:* Größe der Lookup-Table.

Abbildung 5.12 zeigt die Werte der logarithmierten Wahrscheinlichkeiten $\underline{P}_{\kappa_c}$ für verschiedene Festkommadarstellungen bei einer `Fix_5_1` Darstellung der LLRs $L^{\mathrm{i}}(c_k)$. Der Speicherbedarf der Lookup-Table wird dabei durch die Wortbreite pro Eintrag und die nötige Anzahl an Einträgen ermittelt[1]. Numerische Simulationen ergeben, dass eine `uFix_4_1` Darstellung der Werte von $\underline{P}_{\kappa_c}$ für den Max*-Log-MAP Decoder ausreichend ist, um einen nahezu idealen Verlauf der Transinformation zu erhalten. Der Fehler gegenüber einer vollständigen Gleitkommaimplementierung ist dabei kleiner als 10^{-2} (L2-Norm). Für die-

[1] Bsp. 100 Bit für `uFix_5_2` Darstellung → Wortbreite 5 Bit, 20 Einträge

ses Ergebnis wurde das Verhalten und die Leistungsfähigkeit des Decoders simuliert. Die LLRs und die logarithmierten Wahrscheinlichkeiten wurden dazu, wie oben beschrieben, durch Festkommazahlen dargestellt, während die verbleibenden Operationen im Decoder in Gleitkomma umgesetzt wurden. Die Ergebnisse wurden für verschiedene SCCs verifiziert.

Die endgültige Berechnung der Metrik geschieht durch Summation aller logarithmierten Wahrscheinlichkeiten $\underline{P}_{\kappa_c}$ mit $\kappa_c = 0, \ldots, n_c - 1$. Im Falle eines Rate-$\frac{1}{2}$ Codes werden z.B. zwei Werte aus der Lookup-Table addiert. Durch die Addition wird zwar die Anzahl nötiger Nachkommabits nicht beeinflusst, jedoch eventuell ein Überlauf verursacht. Um diesen zu Vermeiden, müssen bei der gewählten `uFix_4_1` Darstellung ($m = 3$) zusätzlich $\lceil \mathrm{ld}(n_c) \rceil$ Bits verwendet werden. Insgesamt werden somit

$$m = 3 + \lceil \mathrm{ld}(n_c) \rceil \tag{5.15}$$

Integerbits für die Festkommadarstellung der $\underline{\gamma}_k(s, s')$ Werte benötigt. Für einen Rate-$\frac{1}{2}$ oder Rate-$\frac{1}{3}$ Code folgt demnach die Darstellung durch `Fix_5_1` bzw. `Fix_6_1`.

5.4.2 Zustandswahrscheinlichkeiten

Zur Bestimmung der Zustandswahrscheinlichkeiten wird die in Abschnitt 4.3.3 beschriebene Forward- und Backward-Recursion verwendet. Für den Max*-Log-MAP Decoder wurde erläutert, dass diese Berechnung mithilfe von (4.36) im Logarithmischen durchgeführt wird. *Bemerkung:* Aufgrund der symmetrischen Berechnung wird im Folgenden lediglich die Forward-Recursion betrachtet. Für die logarithmierten Zustandswahrscheinlichkeiten $\underline{\alpha}_k(s')$ in Vorwärtsrichtung gilt

$$\underline{\alpha}_k(s') = \max^* \left(\underline{\gamma}_k(s_1, s') + \underline{\alpha}_{k-1}(s_1) \,,\, \underline{\gamma}_k(s_2, s') + \underline{\alpha}_{k-1}(s_2) \right) , \tag{5.16}$$

mit den Vorgängerzuständen s_1 und s_2. Zur effizienten Umsetzung von (5.16) in Festkomma werden zwei Eigenschaften berücksichtigt: Zum Einen ist der Operator $\max^*$ linear, zum Anderen ist der Dynamikbereich der $\underline{\alpha}_k(\cdot)$ zu jedem Zeitpunkt k beschränkt.

Linearität von max* Für den Operator folgt nach (4.35) und einer konstanten Verschiebung durch $c \in \mathbb{R}$ [BGG03]

$$\begin{aligned} \max^*(a+c, b+c) &= \max(a+c, b+c) + \ln\left(1 + e^{-|a+c-b-c|}\right) \\ &= c + \max(a+c, b+c) + \ln\left(1 + e^{-|a-b|}\right) \\ &= c + \max^*(a, b)\ . \end{aligned} \tag{5.17}$$

Mithilfe dieser Linearität und der Tatsache, dass der konstante Term c bei der Soft- und Hard-Decision entfällt, ist es möglich, eine Verschiebung aller $\underline{\alpha}_k(s')$ zu jedem Zeitpunkt k um einen Konstante vorzunehmen. Eine dynamische Anpassung der Festkommadarstellung an den wachsenden Wertebereich der Zustandswahrscheinlichkeiten kann somit durch diese *logarithmische* Skalierung vermieden werden.

Beschränkter Dynamikbereich In [BGG03] wurde gezeigt, dass der Dynamikbereich D der $\underline{\alpha}_k(s')$ zu jedem Zeitpunkt k beschränkt ist. Der Bereich kann durch die Ungleichung

$$D \leq \left(2\underline{\gamma}_{\max} + \ln(2)\right)\nu \tag{5.18}$$

abgeschätzt werden, wobei ν der Länge des Schieberegisters im Encoder und $\underline{\gamma}_{\max}$ der größten Metrik entspricht. Da Letztere bereits durch eine Festkommadarstellung begrenzt ist, gilt mit (5.15)

$$\underline{\gamma}_{\max} = 2^{3+\lceil \mathrm{ld}(n_c)\rceil}\ . \tag{5.19}$$

Für die Werte von $\underline{\alpha}_k(s')$ müssen demnach maximal $\lceil \mathrm{ld}(D)\rceil$ Bits verwendet werden bzw. im Vergleich zur Darstellung der Metrik ungefähr $\lceil 1 + \mathrm{ld}(\nu)\rceil$ Bits zusätzlich. Eine effiziente Aufteilung auf Integer- und Nachkommabits wurde simulativ ermittelt. Die Ergebnisse zeigen, dass die zusätzlichen Bits als Nachkommabits zu verwenden sind. Für einen SCC mit Faltungscode $[5,7]_8$ (Metrik: `uFix_5_1`; $\nu = 2$) folgt z.B. die Darstellung `uFix_7_3`. Für einen $[13,15,17;13]_8$-Code (Metrik: `uFix_6_1`; $\nu = 3$) lautet die Darstellung bereits `uFix_9_4`.

Das EXIT-Diagramm in Abbildung 5.13 zeigt den Verlauf der Transinformation zwischen den Codebits c_k und den LLRs $\lambda^D(c_k)$ für einen SCC mit Polynom $[5,7]_8$. Der Max*-Log-MAP Decoder wurde mit

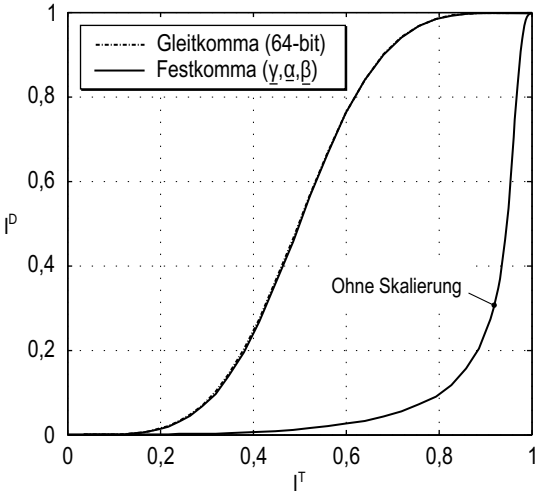


Abb. 5.13: EXIT-Diagramm des Max*-Log-MAP Decoders bei (teilweiser) Implementierung in Festkomma (SCC mit $\mathbf{C} = [5, 7]_8$)

den bereits hergeleiteten Festkommadarstellungen `uFix_5_1` (Metrik) sowie `uFix_7_3` ($\underline{\alpha}_k(s')$, $\underline{\beta}_k(s)$) mit Skalierungsmechanismen implementiert.

Die Ergebnisse verdeutlichen, dass die Darstellungen marginale Verschiebungen der Transinformation verursachen. Der Fehler gegenüber der Gleitkommaimplementierung ist dabei kleiner als 10^{-2} (L2-Norm). Im Falle einer Implementierung ohne Skalierung werden bereits nach wenigen Übergängen im Trellis die Maximalwerte der Festkommadarstellung erreicht und die Konvergenz des Decoders stark beeinträchtigt.

5.4.3 Soft- und Hard-Decision

Als letzter Verarbeitungsschritt folgt die Soft- und Hard-Decision (s. Abs. 4.3.3, Gl. (4.32) und (4.33)), die zur Bestimmung der LLRs $L^o(c_k)$ bzw. $L^o(b_k)$ nötig ist. Letzteres wird durch (4.34) in die geschätzten Infobits $\hat{b}_k$ umgewandelt. In der logarithmierten Version des

MAP-Decoders folgt für die Soft-Decision gemäß (4.37)

$$\begin{aligned} L^{\mathrm{o}}(c_{k'}) = & \max^{*}_{\substack{s \to s' \\ C_{\kappa_c}=1}} \left(\underline{\alpha}_k(s) + \underline{\gamma}_k(s,s') + \underline{\beta}_{k+1}(s') \right) \\ & - \max^{*}_{\substack{s \to s' \\ C_{\kappa_c}=0}} \left(\underline{\alpha}_k(s) + \underline{\gamma}_k(s,s') + \underline{\beta}_{k+1}(s') \right) , \end{aligned} \tag{5.20}$$

wobei alle Übergänge im Trellis von s nach s' auswertet werden, die entweder mit einem Codebit $C_{\kappa_c} = 1$ oder $C_{\kappa_c} = 0$ verknüpft sind. Das *a posteriori* LLR in (5.20) setzt sich aus den extrinsischen LLRs $\lambda^D(c_k)$ und den LLRs am Eingang des SISO-Decoders $\zeta^E(c_k) = L^{\mathrm{i}}(c_k)$ gemäß

$$L^{\mathrm{o}}(c_k) = \lambda^D(c_k) + \zeta^E(c_k) \tag{5.21}$$

zusammen (s. Abb. 4.4). Zur Implementierung der LLRs $\lambda^D(c_k)$ und $\zeta^E(c_k)$ wurden bereits die Darstellung `Fix_5_1` bzw. `Fix_4_0` in Abschnitt 5.2.2 und 5.2.1 bestimmt, sodass es zur Vermeidung von Überläufen zwangsläufig die Festkommadarstellung `Fix_6_1` für die LLRs $L^{\mathrm{o}}(c_k)$ zu verwenden gilt.

Wird der Max*-Log-MAP Decoder zur Decodierung von SCCs eingesetzt, kann eine Vorzeichenentscheidung im Algorithmus zur Hard-Decision getroffen und auf die explizite Darstellung der LLRs $L^{\mathrm{o}}(b_k)$ verzichtet werden. Zur Decodierung von seriell verketten Codes (SCCCs) gelten die gleichen Überlegungen wie für die *a posteriori* LLRs und führen letztendlich auf eine `Fix_6_1` Darstellung der LLRs $L^{\mathrm{o}}(b_k)$.

Tab. 5.6: Festkommadarstellung des Max*-Log-MAP Decoders

Variable	**Darstellung**	**Bemerkung**	**Bsp.** SCC $[5,7]_8$
$\underline{\gamma}_k(s,s')$	`uFix_(m+1)_1`	$m = 3 + \lceil \mathrm{ld}(n_c) \rceil$	`uFix_5_1`
$\underline{\alpha}_k(s)$, $\underline{\beta}_k(s)$	`uFix_(m+n)_n`	$m = 3 + \lceil \mathrm{ld}(n_c) \rceil$ $n = 1 + \lceil 1 + \mathrm{ld}(\nu) \rceil$	`uFix_7_3`
$L^{\mathrm{o}}(c_k)$, $L^{\mathrm{o}}(b_k)$	`Fix_6_1`	/	`Fix_6_1`

Tabelle 5.6 fasst alle hergeleiteten Darstellungen zur Implementierung des Max*-Log-MAP Decoders in Festkommaarithmetik zusammen. Das EXIT-Diagramm in Abbildung 5.14 zeigt den Vergleich zwischen

einer Gleitkomma- und vollständigen Festkommaimplementierung des Decoders für zwei unterschiedliche SCCs.

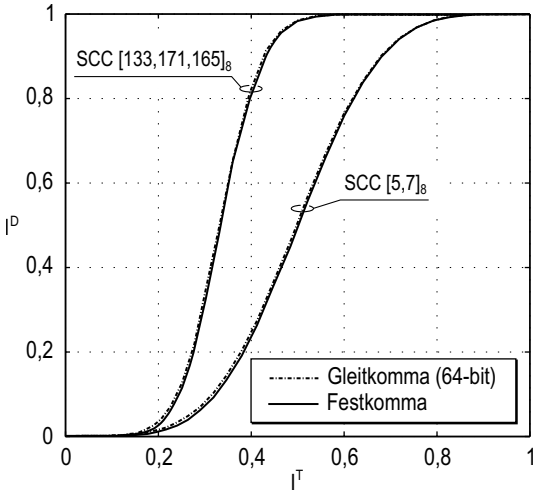


Abb. 5.14: EXIT-Diagramm des Max*-Log-MAP Decoders für eine vollständige Implementierung in Festkomma gemäß Tabelle 5.6

5.4.4 GPP-Implementierung

Obwohl Festkommaarithmetik in erster Linie mit Logiken oder Festkommaprozessoren in Zusammenhang gebracht wird, lässt sich eine Festkommaimplementierung des Max*-Log-MAP Decoders auch auf einem GPP durchführen. Der Decoder wird dazu mit den hergeleiteten Festkommadarstellungen auf einem Intel Core i7 GPP [Ii714] in Festkomma umgesetzt und der Speicherverbrauch sowie die Geschwindigkeit des Algorithmus analysiert. Als Faltungscode wird ein einfacher SCC mit $\mathbf{C} = [5, 7]_8$ verwendet. Die Implementierung erfolgt in C und der Code wird mit dem C-Compiler der GNU Compiler Collection (Version 4.8.1) auf einem Linux Rechner (Ubuntu 13.10) kompiliert.

Da die Programmiersprache C über keinen nativen Festkommadatentyp verfügt, wird auf den kleinsten Integer-Datentyp mit 8-bit (Char)

zurückgegriffen. Dieser wird mithilfe bitweiser Operatoren als eine Art Container für die Festkommadarstellungen des betrachteten Faltungscodes (s. Tab. 5.6) verwendet. Parallel dazu wird der Decoder in einer generischen 32-bit Gleitkommadarstellung (Float Datentyp) umgesetzt und evaluiert. Beide Implementierungen sind *nicht* hinsichtlich ihrer Geschwindigkeit optimiert, sondern unterscheiden sich nur in den verwendeten Datentypen. Die Leistungsfähigkeit der Char-Implementierung deckt sich mit dem prognostizierten Verlauf des SCCs $[5{,}7]_8$ in Abbildung 5.14.

Der Code-Ausschnitt in Listing 5.1 zeigt beispielhaft die Berechnung der Metrik für alle 4 möglichen Codebit-Kombinationen (00, 01, 10, 11) unter Verwendung der in Abschnitt 5.4.1 erläuterten Lookup-Table und Gleichung (5.14). Die LLRs am Eingang $L^{\mathrm{i}}(c_k)$ (Code: `Li`) werden bereits durch einen Bitshift in die Darstellung `Fix_5_0` überführt und entsprechen den 5 Least Significant Bits (LSB) eines Chars. Ebenso belegen die Werte der LUT die unteren 4 Bit des Char Datentyps. Zur Abbildung der LLR-Werte auf die Indices der LUT (Code: `logP_LUT`) wird je nach Codebit das Makro `LUT_idx_1` bzw. `LUT_idx_0` verwendet. Der Wertebereich der so bestimmten Metrik (Code: `Mxx`) liegt zwischen 0 und 30 und beansprucht, wie in Tabelle 5.6 angegeben, die 6 LSBs des Datentyps.

Listing 5.1: Berechnung der Metrik mithilfe der LUT

```
1  #define LUT_idx_1(val) (val+16)
2  #define LUT_idx_0(val) (15-val)
3
4  /* ... */
5
6  static const unsigned char logP_LUT[32] =
7        {15,15,14,13,12,11,10,9,8,7,6,5,4,3,3,2,
8          1, 1, 1, 0, 0, 0, 0,0,0,0,0,0,0,0,0,0};
9
10 /* ... */
11
12 M00 =  logP_LUT[LUT_idx_0(Li[2*k])]
13       +logP_LUT[LUT_idx_0(Li[2*k+1])];
14 M01 =  logP_LUT[LUT_idx_0(Li[2*k])]
15       +logP_LUT[LUT_idx_1(Li[2*k+1])];
16 M10 =  logP_LUT[LUT_idx_1(Li[2*k])]
```

```
17        +logP_LUT[LUT_idx_0(Li[2*k+1])];
18 M11 =   logP_LUT[LUT_idx_1(Li[2*k])]
19        +logP_LUT[LUT_idx_1(Li[2*k+1])];
```

Speicherverbrauch Die Bestimmung des Speicherverbrauchs erfolgt mit den Linux-Hilfsprogrammen `smem` und `pmap`. Der C-Code des Max*-Log-MAP Decoders wird dazu in eine einfache Testapplikation integriert, kompiliert und ausgeführt. Als Bewertungsgröße wird die Proportional Set Size (PSS) des laufenden Prozesses ausgewertet. Die PSS entspricht der Menge an Prozess-eigenem Speicher (*engl.* Unique Memory) plus des proportionalen Anteils an gemeinsamem, mit anderen Prozessen genutztem, Speicher (*engl.* Shared Memory) und spiegelt somit den *wahren* Speicherverbrauch des einzelnen Prozesses wider.

Der Verbrauch der Char- und Float-Implementierung ist in Tabelle 5.7 für verschiedene Infobit-Sequenzlängen N_b zusammengefasst und wurde durch `smem` ermittelt. Die Ergebnisse zeigen, dass die Festkommavariante eine Speicherersparnis erzielt und diese mit der Länge des decodierten Trellis zunimmt. Letzteres ist darin begründet, dass mit N_b auch der allokierte Arbeitsspeicher für Vektoren und Matrizen steigt und dieser ab einer gewissen Größe den Programmspeicher der Testapplikation dominiert. Im hier betrachteten Grenzfall (N_b = 65536) beträgt der Speicherverbrauch gerade noch 27%, was in etwa dem Verhältnis von Char (8-bit) zu Float (32-bit) entspricht. Da der allokierte Arbeitsspeicher exponentiell mit der Länge des Schieberegisters im Encoder wächst, ist für komplexere Faltungscodes bereits für kürzere Infobit-Sequenzlängen mit einer höheren Ersparnis zu rechnen.

Tab. 5.7: Speicherverbrauch und Infobitrate

N_b	**PSS** (kByte)		**Einspar.**	**Infobitrate** (Mbit/s)		**Beschl.**
	Float	**Char**		Float	**Char**	
16	128	98	23%	0,35	24,39	69,69
256	140	98	30%	0,36	25,00	69,44
1024	180	106	41%	0,36	25,00	69,44
65536	3452	926	73%	0,36	25,00	69,44

Neben der reduzierten PSS kann darüber hinaus im Falle der Festkommaimplementierung auf das Laden der Mathematikbibliothek (C Math Library, `libm`) verzichtet werden. Da die rechenaufwendige Logarithmus- und Exponentialfunktion durch eine Lookup-Table realisiert werden, resultiert durch diese Optimierung eine drastische Einsparung von 2,7 MB (ermittelt mit `pmap`).

Geschwindigkeit Die Geschwindigkeit beider Implementierungen wurde mithilfe der POSIX.1b Realtime Extensions Library [Prt93] ermittelt. Diese Bibliothek ermöglicht eine Nanosekunden-genaue Messung von Zeitdifferenzen im Programmcode und ist im Betriebssystem integriert. Die Zeitdauer für einen Decodiervorgang konnte somit bestimmt und mit N_b in eine maximal erzielbare Datenrate umgerechnet werden. Die Ergebnisse in Tabelle 5.7 zeigen, dass durch das Wechseln von Float auf Char eine rund 70-fache Beschleunigung der Max*-Log-MAP Decodierung erreicht wird. Der Geschwindigkeitsgewinn wird erneut durch den Übergang auf die Lookup-Table erzielt, die das Verwenden der rechenaufwendigen Mathematikfunktionen (exp, ln) überflüssig macht.

Zusammenfassung Obwohl die meisten Prozessoren (GPP, ARM, usw.) über eine Gleitkommaarithmetik verfügen und Zahlenwerte mit (fast) beliebiger Genauigkeit darstellen, bietet eine Festkommaimplementierung des Max*-Log-MAP Decoders auf diesen Prozessoren dennoch Vorteile: Neben einem reduzierten Speicherverbrauch kann durch den Verzicht auf rechenaufwendige Mathematikfunktionen ein beachtlicher Geschwindigkeitsgewinn bei gleichbleibender Leistungsfähigkeit des Decoders erzielt werden.

5.5 SCCC-Decoder

Da der SCCC-Decoder aus zwei kombinierten Max*-Log-MAP Decodern besteht, kann auf die Ergebnisse der Festkommaimplementierung aus dem vorherigen Abschnitt zurückgegriffen werden. SCCC-Decoder wurden ebenfalls bereits hinsichtlich ihrer Festkommaaspekte untersucht [YW99][MB00] und effiziente Darstellungen hergeleitet. Die Ergebnisse aus der Literatur decken sich mit denen in dieser Arbeit und sind in Abbildung 5.15 eingezeichnet.

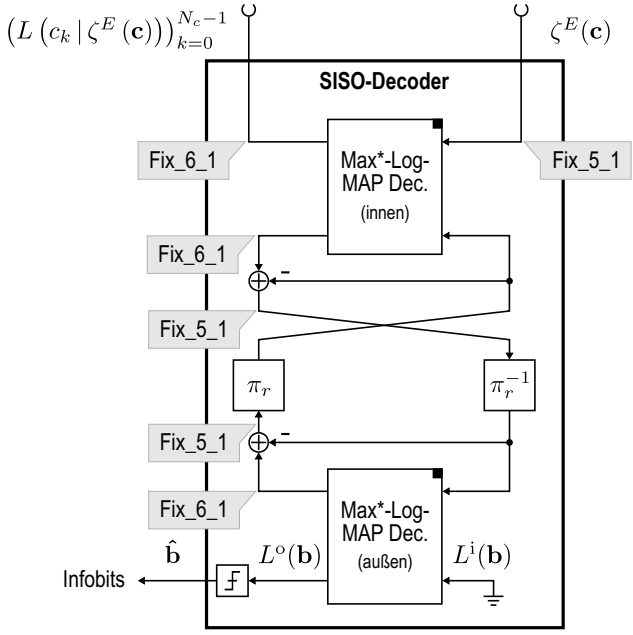


Abb. 5.15: SCCC-Decoder in Festkommadarstellung

Die Leistungsfähigkeit der Festkommaimplementierung ist in Abbildung 4.6 für zwei SCCCs dargestellt. Die Transinformation wird lediglich marginal von der begrenzten Dynamik der Zahlendarstellung beeinflusst und führt auf den gleichen Verlauf wie im Falle einer Umsetzung in Gleitkomma.

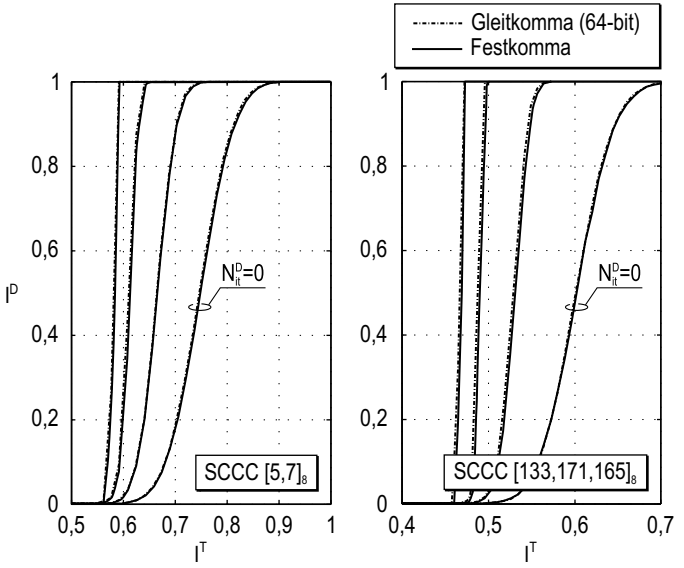


Abb. 5.16: EXIT-Diagramm des SCCC-Decoders für eine vollständige Implementierung in Festkomma gemäß Tabelle 5.6 und Abbildung 5.15 (Iterationen $N_{\text{it}}^{D} = (0; 1; 3; 8)$, innerer Code: $\mathbf{C}_r = [2; 3]_8$).

5.6 Zusammenfassung

Die in diesem Kapitel durchgeführten Analysen zur Umsetzung des SC-MMSE-FD Entzerrers und des Max*-Log-MAP Decoders in Festkomma-Arithmetik dienen zur Implementierung des vollständigen SC-MMSE-FD Turbo-Entzerrers. Die hergeleiteten Modelle und Berechnungsvorschriften erlauben eine effiziente Dimensionierung der Festkommadarstellung aller ressourcenintensiven Symbole bzw. Variablen und können auf beliebige Systemkonfigurationen (Sequenzlängen, Faltungscodes, usw.) angewendet werden. Durch die getrennte Analyse lassen sich die SISO-Komponenten auch auf unterschiedlichen Prozessoren oder Logiken effizient implementieren. Die nötige Darstellung der propagierten extrinsischen Information ist statisch und lautet je nach Richtung `Fix_4_0` bzw. `Fix_5_1`.

Im Falle des SC-MMSE-FD Entzerrers wurden die entstehenden Quantisierungs- und Sättigungsfehler durch stochastische Modelle beschrieben. Mithilfe der Modelle und der ermittelten Qualitätsanforderungen kann die nötige Anzahl an Integer- und Nachkommabits für jedes Symbol im Entzerrer bestimmt werden. Aufgrund der Frequenzbereichsentzerrung steigt der nötige Dynamikbereich der Darstellung beim Übergang vom Zeit- in den Frequenzbereich mit der Länge der durchgeführten DFT. Die Implementierung des SC-MMSE-FD Entzerrers mithilfe einer FPGA-Entwicklungsumgebung zeigen dennoch, dass unter Berücksichtigung der effizienten Darstellungen der Ressourcenverbrauch auf aktuellen FPGAs weniger als 10% beträgt und die Leistungsfähigkeit nur marginal beeinflusst wird.

Für die Untersuchungen des Max*-Log-MAP Decoders konnte partiell auf die Ergebnisse aus der Literatur zurückgegriffen werden. Im Vergleich zum SC-MMSE-FD Entzerrer ist die Festkommadarstellungen der ressourcenintensiven Variablen relativ statisch und lediglich logarithmisch von der Länge des Schieberegisters im Encoder und von der Coderate abhängig. Die Festkommaimplementierung des Decoders auf einem GPP verdeutlicht, dass auf Prozessoren eine drastische Reduzierung des Speicherverbrauchs und eine Beschleunigung des Decodiervorgangs möglich ist. Darüber hinaus wurden die hergeleiteten Darstellungen zur Umsetzung von SCCC-Decodern verifiziert.

KAPITEL 6

Parallelisierbarkeit

In der Informationstechnik werden Vorgänge, die unabhängig voneinander sind und zeitgleich ablaufen, als *parallel* bezeichnet. Des Weiteren gilt diese Bezeichnung meist für Vorgänge, die aus Sicht der Verarbeitung auf gleicher Ebene stattfinden. So werden z.B. Programme parallel ausgeführt oder mathematische Funktionen parallel berechnet. Das Identifizieren und Erzeugen paralleler Vorgänge wird als *Parallelisierung* bezeichnet. Dieses Kapitel beschäftigt sich mit der Parallelisierung der SC-MMSE-FD Turbo-Entzerrung und verdeutlicht anhand von Implementierungen den Nutzen dieser Analysen.

Wie im vorherigen Kapitel ergibt die Literaturrecherche, dass bereits Veröffentlichungen zum Thema Parallelisierung von Turbo-Decodern existieren. Diese entstanden überwiegend im Zusammenhang mit der Einführung von UMTS bzw. LTE, die beide unter Anderem die leistungsfähigen Turbo-Codes zum Fehlerschutz einsetzen. Da Turbo-Decoder und Turbo-Entzerrer als Kernkomponenten MAP-basierte Decoder besitzen, können erneut Synergien ausgenutzt werden. Viele der bereits existierenden Parallelisierungsverfahren müssen dennoch auf die Turbo-Entzerrung angepasst bzw. erweitert werden. Veröffentlichungen, die sich speziell mit der Parallelisierung von (SC-MMSE-FD) Turbo-Entzerren beschäftigen, liegen zum gegenwärtigen Zeitpunkt nicht vor. Im weiteren Verlauf dieses Kapitels wird auf die relevanten Arbeiten in den entsprechenden Abschnitten verwiesen.

Nach einer Einführung in die Grundlagen der Parallelisierung in Abschnitt 6.1 werden die Untersuchungen zunächst für beide SISO-Komponenten des Turbo-Entzerrers getrennt durchgeführt. Abschnitt 6.2 beschäftigt sich aus diesem Grund mit dem SC-MMSE-FD Entzerrer und verdeutlicht mögliche Parallelisierungen. Die Effizienz der Ansätze wird anschließend anhand von Implementierungen auf einem GPP evaluiert und diskutiert. Die Analyse des Max*-Log-MAP Decoders in Abschnitt 6.3 verläuft analog. Im Gegensatz zum Entzerrer wird zusätzlich eine GPU zur Evaluation herangezogen. Den Abschluss dieses Kapitels bildet die Parallelisierung des gesamten SC-MMSE-FD Turbo-Entzerrers in Abschnitt 6.4. Während die Ergebnisse der einzelnen SISO-Komponenten in diese Untersuchung mit einfließen, ermöglicht speziell die Pipeline-Technik auf Iterationsebene eine erhebliche Steigerung der Effizienz. Abschnitt 6.6 fasst alle Ergebnisse zusammen.

6.1 Grundlagen

In Kapitel 2 wurde gezeigt, dass eine parallelisierte Implementierung der Berechnung des Betragsquadrats und der Kreiszahl π die Verarbeitungsgeschwindigkeit erhöht (s. Abs. 2.1.1). Die grundlegende Voraussetzung dafür ist, dass ein Prozessor auch über parallele Verarbeitungseinheiten verfügt. Im Fall der vorgestellten Beispiele wurden dazu die parallelen Vorgänge auf die Prozessor- und Datenebene abgebildet. Die erreichbare Geschwindigkeit der Implementierung ist dabei von der Effizienz der verwendeten Programmierschnittstellen, der Speicherarchitektur und -organisation des Prozessors sowie von der Ausführungsverwaltung (*engl.* Scheduling) des Betriebssystems abhängig. Diese Einflüsse führen zur Erzeugung eines *Overheads*, den es durch gezieltes Parallelisieren und Abbilden zu kompensieren oder gar zu vermeiden gilt.

Anders als bei Prozessoren entsteht beim Abbilden paralleler Vorgänge auf eine Logik kein Overhead. Die Architektur eines FPGAs unterstützt darüber hinaus eine Parallelisierung auf unterschiedlichen Ebenen, da aufgrund der elementaren und unabhängigen Logikgatter eine feine Granularität möglich ist. Der Grad der Parallelisierung ist

in erster Linie durch die Anzahl verfügbarer Gatter bzw. durch die Logikfläche beschränkt. Zusätzliche Einschränkungen ergeben sich durch die Einhaltung der Taktanforderungen (*engl.* Timing Constraints) einer Implementierung [MB07]. Obwohl in diesem Kapitel die Untersuchungen zur Parallelisierung vorrangig im Kontext von Prozessoren durchgeführt werden, sind die Ergebnisse *ohne* Einschränkungen auch auf FPGAs übertragbar.

Formen von Parallelität und Optimierungsziel Parallele Vorgänge in einem Algorithmus können sowohl transparent als auch durch diverse Prozesse maskiert sein. So ist z.B. die Parallelisierung einer Vektoraddition ohne weiteres möglich, während bei einer FFT die Durchführung der Butterlfy-Operationen erst bei genauerer Betrachtung die Möglichkeit zur Parallelisierung bietet. Vorgänge können aber auch aufgrund von Datenabhängigkeiten nicht parallelisierbar sein. In diesem Fall ermöglicht erst die Modifikation des Algorithmus, z.B. der Übergang zu einer suboptimalen Methode, eine *Pseudo*-Parallelität. Ein später noch im Detail erläutertes Beispiel ist die Decodierung einer Codebit-Sequenz. Der rekursive Decodiervorgang der gesamten Sequenz lässt sich in die parallele Decodierung von Teilsequenzen aufteilen. Diese Vorgehensweise ist zwar aus Sicht der erzielbaren Bitfehlerrate suboptimal, ermöglicht aber eine Parallelisierung.

Das Optimierungsziel in diesem Kapitel ist es, eine *effizientere* Auslastung eines Prozessors oder einer Logik durch Parallelisierung zu erreichen. Für Prozessoren müssen dazu unabhängige Ausführungsstränge (*engl.* Threads) erzeugt werden, um das Scheduling des Betriebssystems zu verbessern: Die Aufgabenverteilung auf die parallelen Verarbeitungseinheiten kann mithilfe von Threads ökonomischer und flexibler gestaltet werden. Ist z.B. die Symbolrate der Systemkonfiguration durch einen Funkstandard vorgeben, ermöglicht eine Parallelisierung eventuell eine echtzeitfähige Umsetzung. Gleiches gilt für Logiken, bei denen eine parallelisierte Implementierung die Anzahl an Operation pro Takt erhöht und somit die Verarbeitungsgeschwindigkeit steigert.

Bewertungsgrößen und Grenzen Zur Bewertung einer parallelen Implementierung wurden bereits in Kapitel 2 die Kenngrößen *Beschleunigung* und *Effizienz* eingeführt. Beide Größen beziehen sich auf eine sequentielle Umsetzung und verdeutlichen den Gewinn der Parallelisierung auf einem Prozessor. Ein allgemeines Modell für die erreichbare Beschleunigung wurde 1967 durch Amdahl beschrieben [Amd67]. Danach teilt sich die Laufzeit eines Programms in einen parallelisierbaren Anteil ρ und einen sequentiellen Anteil $1 - \rho$ auf. Während ρ auf K unabhängige Verarbeitungseinheiten aufteilt werden kann und ein Geschwindigkeitszuwachs entsteht, bleibt der sequentielle Anteil konstant. Für die Beschleunigung folgt demnach mit (2.1)

$$B_K = \frac{1}{1 - \rho(1 - \frac{1}{K})} \,. \tag{6.1}$$

Das Amdahl'sche Gesetz besagt, dass im Grenzfall $K \to \infty$ die Beschleunigung durch den sequentiellen, nicht parallelisierbaren Anteil beschränkt ist. Sind z.B. beide Anteile gleich groß, folgt laut Amdahl eine maximale Beschleunigung von $\lim_{K \to \infty} B_K = 2$.

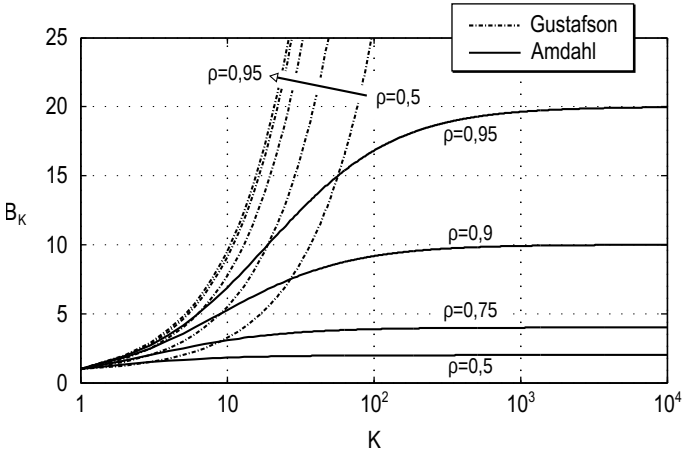


Abb. 6.1: Beschleunigung nach Amdahl und Gustafson für $\rho \in \{0{,}5; 0{,}75; 0{,}9; 0{,}95\}$.

Amdahls Gesetz stellt eine vereinfachte Betrachtungsweise der Be-

schleunigung auf Prozessoren dar [HL09]. So wird außer Acht gelassen, dass mit der Anzahl an unabhängigen Verarbeitungseinheiten meist auch die Größe des schnellen Pufferspeichers (*engl.* Cache) wächst. Dies kann unter günstigen Umständen zu einer *super-linearen* Beschleunigung ($B_K > K$) führen. Des Weiteren wird die bereits erläuterte Effizienz der Programmierschnittstellen, des Speichers und des Schedulings nicht im Modell berücksichtigt. Alternative Modelle, wie das Gustafson'sche Gesetz [Gus88], liefern zwar eine optimistischere Abschätzung, sind aber nicht auf die Problemgröße der Signalverarbeitung übertragbar, die in dieser Arbeit betrachtet wird. Die Parallelisierung der SC-MMSE-FD Turbo-Entzerrung wird zeigen, dass das Amdahl'sche Gesetz in diesem Fall eine *obere* Grenze der Beschleunigung darstellt. Abbildung 6.1 zeigt den Verlauf der Beschleunigung für beide Gesetze und verschiedene parallele Anteile.

6.2 SC-MMSE-FD Entzerrer

Zur Parallelisierung des SC-MMSE-FD Entzerrers wird zunächst eine Ressourcenanalyse des Verfahrens durchgeführt. Sowohl für Prozessoren als auch für Logiken ist dabei die Anzahl der nötigen Multiplikationen entscheidend, da diese im Gegensatz zu einfachen arithmetischen Operationen wie der Addition oder Subtraktion in ihrer Komplexität dominieren. Es ist somit möglich, die relative Verarbeitungsdauer einzelner Schritte des Entzerrers abzuschätzen. Für die Analyse werden lediglich die Symbole bzw. Variablen betrachtet, die einen signifikanten Ressourcenverbrauch verursachen. Ebenso gelten die folgenden Tatsachen:

- Eine Radix-2 N-FFT benötigt $\frac{N}{2}\mathrm{ld}\left(\frac{N}{2}\right)$ komplexe Multiplikationen [KK06, S. 238].
- Eine komplexe Multiplikation erfordert drei *reelle* Multiplikationen[1].

[1] Berechnung: $(a+jb)(c+jd) = \underbrace{ac}_{1} - \underbrace{bd}_{2} + j(\underbrace{(a+b)(c+d)}_{3} - \underbrace{ac}_{1} - \underbrace{bd}_{2})$

Unter Berücksichtigung dieser Vorgaben kann der Verbrauch der Entzerrung (s. Gl. (4.10)) ermittelt werden. Tabelle 6.1 zeigt die Ergebnisse der Analyse und Abbildung 6.2 die dazugehörigen Code-Sektionen sowie Bezeichnungen. Es wird deutlich, dass ein Großteil der Multiplikationen zur Durchführung der FFT benötigt wird. Im Beispiel für $N_s = 1024$ sind es rund 70 %. Aus diesem Grund wird im Weiteren zuerst die Optimierung der FFT erläutert und anschließend auf die Parallelisierung der elementweisen Operationen und des Algorithmus eingegangen.

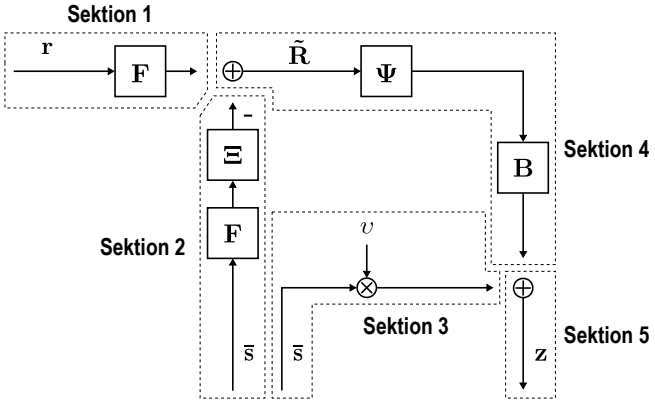


Abb. 6.2: Sektionen des SC-MMSE-FD Entzerrers

6.2.1 Schnelle Fourier-Transformation

Die schnelle Fourier-Transformation (*engl.* Fast Fourier Transform, FFT) ist eine der wichtigsten Operationen der digitalen Signalverarbeitung. Seit der (ersten) Veröffentlichung[1] von Cooley und Tukey [CT65] aus dem Jahr 1965 sind zahlreiche Arbeiten erschienen, die sich im Detail mit der Implementierung dieser sogenannten Radix-2 FFT beschäftigen oder alternative Berechnungsformen (z.B. Radix-4/8, Winograd-Algorithmus, Primfaktor-Algorithmus) vorschlagen.

[1] Die Urform des Algorithmus geht auf Carl Friedrich Gauß zurück [HJB84].

Tab. 6.1: Ressourcenverbrauch

Berechnung	**Anzahl reeller Mult.**	**Bsp.:** $N_s = 1024$
φ (s. Gl. (4.8))	$2N_s$	2048 (3,5 %)
υ (s. Gl. (4.11))	$2N_s$	2048 (3,5 %)
$\mathbf{\Psi}$ (s. Gl. (4.10))	$5N_s$	5120 (8,7 %)
Sektion 1	$1{,}5\,N_s \mathrm{ld}\left(\frac{N_s}{2}\right)$	13824 (23,5 %)
Sektion 2	$3N_s + 1{,}5\,N_s \mathrm{ld}\left(\frac{N_s}{2}\right)$	16896 (28,7 %)
Sektion 3	$2N_s$	2048 (3,5 %)
Sektion 4	$3N_s + 1{,}5\,N_s \mathrm{ld}\left(\frac{N_s}{2}\right)$	16896 (28,7 %)
Sektion 5	-	-
Insgesamt	$17N_s + 4{,}5\,N_s \mathrm{ld}\left(\frac{N_s}{2}\right)$	**58880 (100 %)**
FFTs	$4{,}5\,N_s \mathrm{ld}\left(\frac{N_s}{2}\right)$	41472 (70,4 %)
Elementw. Op.	$17N_s$	17408 (29,6 %)

Die zentralen Optimierungen zur Ausführung der schnellen Fourier-Transformation auf Prozessoren oder Logiken sind heutzutage in leistungsfähigen Software-Bibliotheken bzw. in IP-Cores integriert. Bei der in dieser Arbeit verwendeten FFTW (Fastest Fourier Transform in the West) [FFT14] handelt es sich um eine Implementierung, die speziell für x86-basierte Prozessoren optimiert wurde und unter der GNU General Public License [GPL14] frei verfügbar ist. FFTW adaptiert sich auf das jeweilige System und verwendet eine Kombination aus Radix-2, Rader-Algorithmus und Primfaktor-Algorithmus [FJ05]. Neben der Parallelität auf Prozessorebene nutzt die Software zusätzlich gezielt den SIMD-Befehlssatz des Prozessors [FJ05]. *Zum Vergleich:* Die Ausführung einer nicht optimierten Radix-2 FFT-Implementierung auf einem Intel Core i7 GPP [Ii714] benötigt 461 ns für 1024 komplexwertige Eingangswerte. Mithilfe der FFTW kann die Berechnung auf 1 ns verringert werden. Weitere Vergleiche finden sich in [FJ05].

Zur Ausführung der FFT auf FPGAs existieren, wie bereits in Abschnitt 5.3.3 verwendet, IP-Cores von unterschiedlichen Herstellern und Open Source Gruppen [OCo14]. Der hier verwendete proprietäre FFT IP-Core von Xilinx [XIP14] nutzt eine speziell für Logiken geeignete Pipeline-Technik sowie FPGA-spezifische Optimierungen.

Aufgrund der erläuterten Verfügbarkeit hochoptimierter Bibliotheken und IP-Cores mit unterschiedlichen Parallelisierungstechniken ist von einer eigenhändigen Optimierung der FFTs im SC-MMSE-FD Entzerrer abzusehen. Stattdessen wird in dieser Arbeit der Fokus auf die Parallelisierung der elementweisen Operationen und des Algorithmus selbst gelegt.

6.2.2 Elementweise Operationen

Die neben der schnellen Fourier-Transformation verbleibenden arithmetischen Operationen im Entzerrer können nach Übergang in den Frequenzbereich elementweise auf die Vektoren angewendet werden. Ein Beispiel ist die eigentliche Entzerrung $\mathbf{\Psi}\tilde{\mathbf{R}}$ (s. Gl. (4.10)). Da es sich bei $\mathbf{\Psi}$ um eine Diagonalmatrix handelt, können die einzelnen Elemente der Diagonalen und die Symbole des Vektors $\tilde{\mathbf{R}}$ unabhängig voneinander und zeitgleich über die gesamte Blocklänge N_s miteinander multipliziert werden. Da der Anteil dieser Operationen am Gesamtaufwand nach erster Abschätzung rund 30% (N_s = 1024, s. Tab. 6.1) beträgt, besteht an dieser Stelle ein Optimierungs- bzw. Parallelisierungspotenzial.

Zur Ausnutzung dieser Parallelität, die aus Sicht der Signalverarbeitung auf unterster arithmetischer Ebene besteht, eignen sich speziell die in Abschnitt 2.1.1 erläuterten SIMD-Befehlssätze von Prozessoren. Die Evaluation dieser Parallelisierung auf Datenebene mithilfe von SIMD wird in Abschnitt 6.2.4 durchgeführt.

Auf Logiken können elementweise Operationen aufgrund deren Architektur direkt nebenläufig implementiert werden. Einzig und allein die Anzahl zeitgleich nutzbarer Logikgatter und dedizierter Gatter, wie z.B. Multiplizierer, limitieren hier den Grad der Parallelisierung.

6.2.3 Algorithmus

Mit Blick auf das Blockschaltbild des SC-MMSE-FD Entzerrers in Abbildung 6.2 und den Abhängigkeitsgraphen in Abbildung 6.3 wird

deutlich, dass sich der Algorithmus in drei sequentielle Verarbeitungsstufen unterteilen lässt:

1. Sektion 1 und 2 sowie die Matrix $\mathbf{\Psi}$ können in der ersten Stufe parallel berechnet werden, da sie keine Datenabhängigkeiten untereinander aufweisen. Im Kontext der Turbo-Entzerrung ist dabei zu beachten, dass in der nullten Iteration die Berechnung von Sektion 2 aufgrund von $\bar{s}_k = 0 \; \forall k$ unnötig ist. Ebenso ist für die folgenden Iterationen eine erneute Ausführung von Sektion 1 redundant, da das Ergebnis unabhängig von den Iterationen ist.

2. Liegen die Ergebnisse der ersten Verarbeitungsstufe vor, können Sektion 3 und Sektion 4 parallel ausgeführt werden. Auch hier gilt, dass in der initialen Entzerrung die Berechnung von Sektion 3 entfällt.

3. In der letzten Stufe wird die Addition der Ergebnisse aus Sektion 3 und 4 in Sektion 5 durchgeführt.

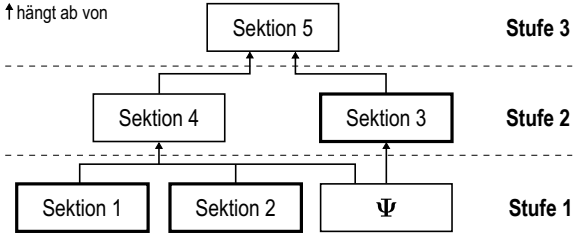


Abb. 6.3: Abhängigkeitsgraph des Algorithmus (SC-MMSE-FD Entzerrer)

Diese Form der Nebenläufigkeit, die verglichen mit den elementweisen arithmetischen Operationen auf einer höheren Ebene existiert, lässt sich auf Prozessorebene ausnutzen und wird anhand einer GPP-Implementierung in Abschnitt 6.2.4 evaluiert. Im Fall von Logiken lassen sich die parallelen Berechnungen in den einzelnen Verarbeitungsstufen als unabhängige Verarbeitungszweige umsetzen.

6.2.4 GPP-Implementierung

Die Evaluation des bereits erläuterten Parallelisierungspotentials erfolgt mithilfe eines Intel Core i7 GPPs [Ii714]. Der SC-MMSE-FD Entzerrer wird dazu in C implementiert und der Code mit dem C-Compiler der GNU Compiler Collection (Version 4.8.1) auf einem Linux Rechner (Ubuntu 13.10) kompiliert. Zur Ausnutzung der SIMD-Rechenarchitekturen wird unter Anderem das GNU Radio Subprojekt VOLK (Vector-Optimized Library of Kernels) [VOL14] verwendet. Zur Thread-Programmierung kommt OpenMP (Open Specifications for Multi Processing) [OMP14] zum Einsatz. Die Details der Umsetzung sowie die Ergebnisse der Parallelisierung der elementweisen Operationen und des Algorithmus werden im Folgenden beschrieben.

Elementweise Operationen VOLK ist eine Prozessor-unabhängige Programmierschnittstelle und verfügt über eine Bibliothek optimierter Funktionen (Kernels), die den SIMD-Befehlssatz des Prozessors verwenden [VOL14][RMO13]. Da ein Prozessor über unterschiedliche Befehlssätze verfügt (z.B. SSEn, SSE4.1, SSE4,2, AVX, usw.), ermittelt VOLK in einem einmaligen Testlauf die schnellste Implementierung für jeden Kernel. Funktionen, die noch nicht in VOLK vorhanden sind, werden durch native SIMD-Intrinsics auf dem GPP umgesetzt.

Die Implementierung des Entzerrers wird für die Blocklängen $N_s \in \{512; 1024; 2048; 4096\}$ vorgenommen und die Beschleunigung durch die SIMD-Architektur untersucht. Dabei werden alle verarbeiteten Symbole durch 32-bit Float-Datentypen repräsentiert. Unter Berücksichtigung der in Abschnitt 5.3 ermittelten Festkommadarstellung wäre es auch möglich Datentypen mit geringerer Wortbreite zu verwenden. So ist für eine Länge von $N_s = 1024$ und eine 16-QAM eine Darstellung durch 16-bit ausreichend (s. Tab. 5.3). Der Vorteil einer geringeren Wortbreite liegt darin, dass zeitgleich mehr Symbole während einer Instruktion in der ALU des Prozessors verarbeitet werden können. Da jedoch keiner der durch den i7 Prozessor unterstützten SIMD-Befehlssätze über native 16-bit Multiplizierer verfügt, wird der Float-Datentyp als Kompromisslösung gewählt.

Listing 6.1 zeigt exemplarisch die Umsetzung der MMSE-Frequenzbereichsentzerrung mithilfe des AVX-Befehlssatzes. Da ein AVX-Register eine Breite von 256-bit besitzt, werden jeweils *acht* Float-Eingangswerte parallel verarbeitet. Nach dem Laden der Register wird die komplexe Multiplikation mittels reellen Multiplikationen, Additionen und Subtraktionen ausgeführt. Das dazu nötige Permutieren der Registerwerte verursacht dabei einen gewissen Overhead.

Listing 6.1: Parallelisierung auf Datenebene (SIMD Code-Schnipsel)

```
1  void simd_cmult_AVX(float complex*       Z_prime,
2                      const float complex* R_tilde,
3                      const float complex* Psi,
4                      unsigned int         Ns) {
5
6    const unsigned int eighthPoints = Ns/8;
7
8    float* Z = (float*)Z_prime;
9    const float* R = (float*)R_tilde;
10   const float* P = (float*)Psi;
11
12   __m256 R1, R2, P1, P2, Z1, Z2, tmp1, tmp2;
13   __m256i mask = _mm256_set_epi32(6,7,4,5,2,3,0,1);
14
15   for(unsigned int k; k < eighthPoints; k++) {
16     /* Beschreiben der AVX Register */
17     R1   = _mm256_load_ps(R); R += 8;
18     R2   = _mm256_load_ps(R); R += 8;
19     P1   = _mm256_load_ps(P); P += 8;
20     P2   = _mm256_load_ps(P); P += 8;
21     /* Multiplikation (Realteil von Z') */
22     tmp1 = _mm256_mul_ps(R1,P1);
23     tmp2 = _mm256_mul_ps(R2,P2);
24     /* Subtraktion (Realteil von Z') */
25     Z1   = _mm256_hsub_ps(tmp2,tmp1);
26     P1   = _mm256_permutevar_ps(P1, mask);
27     P2   = _mm256_permutevar_ps(P2, mask);
28     /* Multiplikation (Imaginaerteil von Z') */
29     tmp1 = _mm256_mul_ps(R1,P1);
30     tmp2 = _mm256_mul_ps(R2,P2);
31     /* Addition (Imaginaerteil von Z') */
```

```
32      Z2   = _mm256_hadd_ps(tmp2,tmp1);
33      tmp1 = _mm256_unpackhi_ps(Z1, Z2);
34      tmp2 = _mm256_unpacklo_ps(Z1, Z2);
35      /* Auslesen der AVX Register */
36      _mm256_store_ps(Z,tmp1); Z += 8;
37      _mm256_store_ps(Z,tmp2); Z += 8;
38    }
39  }
```

Die Ergebnisse der Parallelisierung der elementweisen Operationen sind in Tabelle 6.2 zusammengefasst. Zur Berechnung der Fourier-Transformationen wurde die FFTW-Bibliothek verwendet. Zunächst ist festzustellen, dass für alle untersuchten Blocklängen eine rund 6-fache Beschleunigung durch Ausnutzung der SIMD-Befehlssätze entsteht. Dass diese nahezu konstant bzw. unabhängig von N_s ist zeigt, dass durch SIMD kein konstanter Overhead erzeugt wird. Dieser würde sich mit steigender Blocklänge zugunsten der Beschleunigung amortisieren. Mithilfe der SIMD-Architekturen ist somit eine effiziente Umsetzung der elementweisen Operationen möglich. Da die SIMD-Architektur von DSPs und ARMs ähnlich zu denen eines GPPs ist, sind die Ergebnisse dieser Untersuchung qualitativ übertragbar.

Tab. 6.2: Ergebnisse der Parallelisierung mithilfe von SIMD-Befehlssätzen

N_s	**Symbolrate** (MHz)		**Beschleunigung**
	Sequentiell	**Parallel**	
512	16,53	105,29	6,4
1024	16,39	102,04	6,2
2048	16,26	97,09	6,0
4096	15,85	85,47	5,4

Des Weiteren wird die *relative* Berechnungsdauer der elementweisen Operationen durch Verwendung der FFTW erhöht und somit der Parallelisierungsgewinn gesteigert. Zur Verdeutlichung dieser Form der Selbstverstärkung dient das folgende Rechenbeispiel mit $N_s = 1024$: Die theoretische Ressourcenanalyse des SC-MMSE-FD Entzerrers ergibt für diese Länge eine Aufteilung von 29,6 % für die elementweisen Operationen und 70,4 % für die FFTs (s. Tab. 6.1). Wird nun angenommen, dass eine reelle Multiplikation eine konstante Rechenzeit

beansprucht und die FFTs durch konventionelle, nicht parallelisierte Radix-2 Algorithmen implementiert werden, folgt für die zu erwartende Beschleunigung nach Amdahl (6.1)

$$B_K = \frac{1}{0{,}704 - \frac{0{,}296}{K}} \,. \tag{6.2}$$

Bei $K = 8$ unabhängigen Verarbeitungseinheiten, was einer reinen Implementierung durch AVX (256-bit Register) und 32-bit Float-Datentypen entspricht, folgt somit eine *theoretisch* erreichbare Beschleunigung von nur 1,5. Kann jedoch die relative Dauer des sequentiellen Anteils durch hochoptimierte FFT-Bibliotheken (z.B. FFTW) reduziert werden, sind höhere Gewinne durch die Parallelisierung mithilfe der SIMD-Befehlssätze möglich. Die Ergebnisse in Tabelle 6.2 belegen dieses Verhalten.

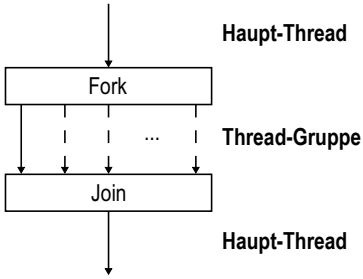


Abb. 6.4: Fork-Join Modell

Algorithmus OpenMP ist ein offener Standard zur Programmierung von GPPs mit gemeinsamem Hauptspeicher (*engl.* Shared Memory) [OMP14][HL09], wie z.B. Mehrkernprozessoren. Die Schnittstelle arbeitet nach dem in Abbildung 6.4 dargestellten klassischen *Fork-Join* Modell und übernimmt die Aufteilung (Fork), die Verwaltung und das Zusammenführen (Join) der Threads. Die Abbildung Thread→Prozesorkern wird im Gegensatz dazu vom Scheduler des Betriebssystems vorgenommen. Die Umsetzung des Modells erfolgt durch primitive Steueranweisungen im Programmcode (Pragma-Direktiven). Listing 6.2 zeigt die prinzipielle Parallelisierung der ersten Verarbeitungsstufe mithilfe von OpenMP's `parallel sections` [HL09].

Listing 6.2: Parallelisierung der ersten Verarbeitungsstufe (OpenMP Code-Schnipsel)

```
1  /* Haupt-Thread */
2  #pragma omp parallel sections        /* <== Fork */
3  {
4    #pragma omp section
5    {
6      /* Sektion 1 (nur in der 0ten Iteration) */
7    }
8    #pragma omp section
9    {
10     /* Sektion 2 (nicht in der 0ten Iteration) */
11   }
12   #pragma omp section
13   {
14     /* Berechnung von Phi */
15   }
16 }                  /* implizite Barriere <== Join */
17 /* Haupt-Thread */
```

Tabelle 6.3 enthält die Ergebnisse der Parallelisierung des Algorithmus. Der Gewinn durch die Aufteilung auf unabhängige Prozessorkerne liefert lediglich eine Beschleunigung von rund 1,2. Die Ursache hierfür liegt im Overhead, der durch den *Kontextwechsel* verursacht wird [SNJ12]. Als Kontext wird der mit einem Thread assoziierte Programm- und Datenspeicher bezeichnet. Der Wechsel des Kontexts zwischen Prozessorkernen geschieht nach dem Fork und vor dem Join und erfordert zeitaufwendige Lese- und Schreibzugriffe auf den Speicher. Da demgegenüber die eigentliche Bearbeitungszeit einer Sektion für die hier betrachteten Blocklängen gering ist, kann der Parallelisierungsaufwand nicht kompensiert werden.

Hinzukommend ist der Signalverarbeitungsaufwand pro Thread während einer Verarbeitungsstufe nicht ausgeglichen (s. Tab. 6.1). So ist z.B. für Sektion 4 die rund 8-fache Anzahl an reellen Multiplikation (N_s = 1024, s. Tab. 6.1) gegenüber Sektion 3 durchzuführen. Obwohl sich diese Situation in der Praxis erneut mit der FFTW verbessern lässt, ist der Parallelisierungsgewinn dennoch reduziert.

Tab. 6.3: Ergebnisse der Parallelisierung auf Prozessorebene

N_s	**Symbolrate** (MHz)		**Beschleunigung**
	Sequentiell	**Parallel**	
512	16,53	19,80	1,2
1024	16,39	20,00	1,2
2048	16,26	19,96	1,2
4096	15,85	19,96	1,3

Zusammenfassend eignet sich die Parallelisierung des Algorithmus nur geringfügig zur Steigerung der Leistungsfähigkeit auf Mehrkernprozessoren. Eine effizientere Möglichkeit die Auslastung von Mehrkernprozessoren zu steigern ist das in Abschnitt 6.4 erläuterte Pipelining.

6.3 Max*-Log-MAP Decoder

Der Max*-Log-MAP Decoder ist die zweite SISO-Komponente im Turbo-Entzerrer, die in diesem Kapitel hinsichtlich Nebenläufigkeiten untersucht wird. Aus Sicht der Parallelisierung kann die Analyse in die Ebenen

- elementare Operationen,
- Algorithmus und
- Subtrellis-Decodierung

aufgeteilt werden. Speziell die in der Einleitung erwähnte Erzeugung einer Pseudo-Parallelität durch die Subtrellis-Decodierung ermöglicht hohe Parallelisierungsgewinne. Darüber hinaus ist die Identifizierung existierender Parallelität Gegenstand der Untersuchungen. Die Ergebnisse der Evaluation auf einem GPP und einer GPU heben erneut die Bedeutung dieser Analysen hervor.

6.3.1 Elementare Operationen

Die elementaren Operationen sind in Abschnitt 4.3.4 für den Max*-Log-MAP Decoder detailliert beschrieben und umfassen die Berechnung der Metrik, der Zustandswahrscheinlichkeiten und der Soft- und Hard-Decision. Im Folgenden werden die Parallelisierungsmöglichkeiten der einzelnen Operationen aufgezeigt.

Metrik Die Übergangswahrscheinlichkeit $\underline{\gamma}_k(s, s')$ von Zustand s nach s' zum Zeitpunkt k wird gemäß Gleichung (5.14) bestimmt und ist lediglich von denjenigen LLRs $L^{\mathrm{i}}(c_{n_c k+\kappa_c})$ abhängig, für die dieser Übergang ausgewertet wird (s. Abb. 4.5). Für eine Infobit-Sequenz der Länge N_b und ein Schieberegister der Länge ν im Encoder lassen sich demnach alle $N_b \cdot 2^{1+\nu}$ Metriken im Trellis parallel berechnen.

Zustandswahrscheinlichkeiten Mithilfe der Forward- und Backward-Recursion (4.36) werden die Zustandswahrscheinlichkeiten $\underline{\alpha}_k(\cdot)$ und $\underline{\beta}_k(\cdot)$ rekursiv durch den gesamten Trellis bestimmt (s. Abb. 4.5). Wie der Name jedoch andeutet, ist aufgrund der Datenabhängigkeit von den zuvor berechneten Werten $\underline{\alpha}_{k-1}(\cdot)$ bzw. $\underline{\beta}_{k+1}(\cdot)$ eine Parallelisierung dieser Berechnung in Zeitrichtung k nicht möglich. Nichtsdestotrotz ist zu einem *festen* Zeitpunkt der Forward-Recursion die Bestimmung der Wahrscheinlichkeiten $\underline{\alpha}_k(0), \ldots, \underline{\alpha}_k(2^\nu - 1)$ unabhängig voneinander und somit für alle 2^ν Zustände parallel durchführbar. Diese sogenannte State-Level Parallelisierung ist aufgrund der symmetrischen Berechnungsvorschrift auch auf die Backward-Recursion übertragbar.

Soft- und Hard-Decision Die Berechnung der *a posteriori* LLRs $L^{\mathrm{o}}(\mathbf{c})$ am Ausgang des Decoders erfolgt durch Gleichung (4.37). Als Voraussetzung für die Parallelisierbarkeit müssen die Zustandswahrscheinlichkeiten und Metriken für den gesamten Trellis bekannt sein. Ist dies der Fall, können alle N_c LLRs zeitgleich berechnet werden. Die Berechnung der LLRs $L^{\mathrm{o}}(\mathbf{b})$ lässt sich analog dazu für alle N_b Werte parallelisieren.

6.3.2 Algorithmus

Die Nebenläufigkeiten im Algorithmus lassen sich erneut mithilfe eines Abhängigkeitsgraphen identifizieren. Abbildung 6.5 zeigt die Zusammenhänge und verdeutlicht, dass lediglich in Stufe 2 die Forward- und Backward-Recursion zur Bestimmung der Zustandswahrscheinlichkeiten parallel berechnet werden. Die $N_b \cdot 2^{1+\nu}$ Berechnungen pro Rekursion können somit nach Bestimmung aller Metriken parallel ausgeführt werden.

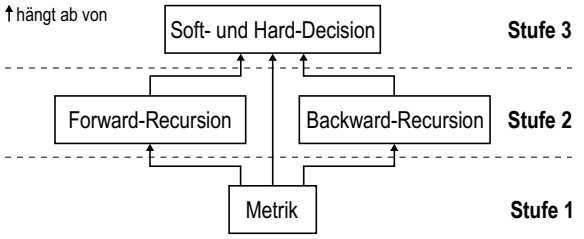


Abb. 6.5: Abhängigkeitsgraph des Algorithmus (Max*-Log-MAP Decoder)

In Abschnitt 6.3.1 wurde erläutert, dass die Berechnung der Info- und Codebit-LLRs zeitgleich geschehen kann, sobald die Forward- und Backward-Recursion, respektive alle Zustandswahrscheinlichkeiten, bekannt sind. Da dies nach $N_b/2$ Iterationsschritten für beide Rekursionen zutrifft, können bereits ab der *Mitte* des Trellis die LLRs *fortlaufend* zur Forward- und Backward-Recursion bestimmt werden. Diese Form der Parallelisierung stellt eine Aufweichung der diskreten Stufengrenzen in Abbildung 6.5 dar und wird in Abschnitt 6.3.4 zur Implementierung verwendet. Abbildung 6.6 verdeutlicht das Vorgehen.

6.3.3 Subtrellis-Decodierung

Nach der Parallelisierung der elementaren Operationen und des Algorithmus stellt die Forward- und Backward-Recursion mit je N_b sequentiell durchzuführenden Berechnungen den Flaschenhals der Decodierung dar. Um diese Berechnungen dennoch zu parallelisieren, wird in

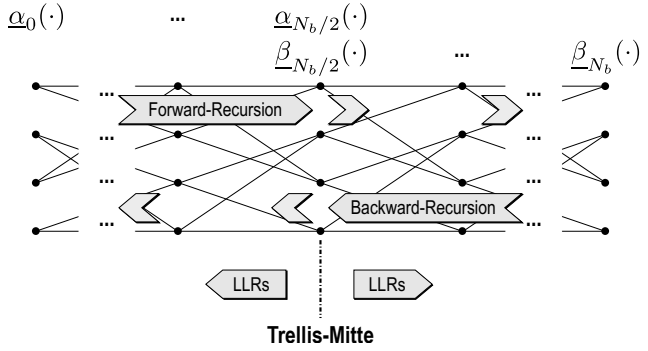


Abb. 6.6: Schematische Darstellung der parallelen Berechnung der LLRs und der Forward- bzw. Backward-Recursion

dieser Arbeit die suboptimale Subtrellis-Decodierung verwendet. Die Decodierung der gesamten LLR-Sequenz $L^{\mathrm{i}}(\mathbf{c})$ der Länge N_c wird dazu in die Decodierung von Teilsequenzen zerlegt [ZP04][MBJ06][SJ13]. Eine Teilsequenz besitzt die Länge $N_{c,s}$, mit $0 < N_{c,s} \leq N_c$, die ein ganzzahliges Vielfaches der Anzahl an Codebits pro Trellis-Übergang n_c ist. Auf Basis eines Subtrellis der Länge $N_{b,s} = N_{c,s}/n_c$ können somit $K_s = N_c/N_{c,s}$ Decodierungen *parallel* ausgeführt werden. Die Teilergebnisse der einzelnen Subdecoder werden anschließend wieder zusammengeführt und ergeben die LLRs der Codebits $L^{\mathrm{o}}(\mathbf{c})$ und Infobits $L^{\mathrm{o}}(\mathbf{b})$.

Der Gewinn der erzeugten Pseudo-Parallelität wird jedoch durch ein vermindertes Leistungsvermögen der Max*-Log-MAP-Decodierung erkauft. Bis jetzt wurde angenommen, dass die Zustandswahrscheinlichkeiten zu Beginn und am Ende des Trellis zur Berechnung der Forward- und Backward-Recursion bekannt sind. Dies wurde mittels eines definierten Startzustands des Encoders sowie einer Terminierung erreicht und ist auch weiterhin gültig. Für die Subtrellis-Decodierung ist diese Voraussetzung jedoch für einen *einzelnen* Subtrellis nicht mehr erfüllt. Die Zustandswahrscheinlichkeiten an den Grenzen müssen deshalb durch suboptimale Ansätze bestimmt werden. Dies reduziert die Zuverlässigkeit der Info- und Codebit-LLRs und führt letztendlich zu einer erhöhten Bitfehlerrate. Im Folgenden werden zwei Methoden be-

schrieben, die Zustandswahrscheinlichkeiten an diesen Grenzen zu bestimmen.

Direkte Methode Die nächstliegende Methode die unbekannten Zustandswahrscheinlichkeiten an den Grenzen eines Subtrellis zu bestimmen, ist die Zustände als *gleichwahrscheinlich* anzunehmen. Die Initialisierung der Forward- und Backward-Recursion geschieht demnach für jeden Subdecoder durch

$$\begin{aligned} \underline{\alpha}_0(s) &= \ln\left(2^{-\nu}\right), \quad s = 0, \ldots, 2^{\nu} - 1 \\ \underline{\beta}_{N_{b,s}}(s) &= \ln\left(2^{-\nu}\right). \end{aligned} \tag{6.3}$$

Während die Implementierung dieser direkten Methode unkompliziert ist, reduziert sich jedoch das Leistungsvermögen jedes einzelnen Subdecoders und somit auch der gesamten Decodierung. In Abbildung 6.7 ist das EXIT-Diagramm einer Subdecodierung mit direkter Methode für einen einfachen Faltungscode $\mathbf{C} = [5, 7]_8$ dargestellt. Die zu decodierende LLRs-Sequenz $L^{\mathrm{i}}(\mathbf{c})$ der Länge $N_c = 2048$ wurde dazu einmal in $K_s = 128$, 32 und 4 Teilsequenzen aufgeteilt und parallel decodiert. Zusätzlich ist zum Vergleich der Verlauf für eine konventionelle Max*-Log-MAP Decodierung ohne Parallelisierung eingezeichnet.

Diese Kurven verdeutlichen, dass sich mit steigender Subtrellis-Länge $N_{b,s}$ der Fehler durch die *fälschliche* Annahme gleichwahrscheinlicher Zustände bei der Initialisierung der Rekursionen reduziert. Für den hier betrachteten Faltungscode ist bereits für eine Subtrellis-Länge von 256 mit einer nahezu perfekten Leistungsfähigkeit der Subtrellis-Decodierung zu rechnen. Entsprechende Ergebnisse wurden für diverse andere Faltungscodes, mit teils rekursiver und komplexerer Code-Struktur, beobachtet. Zu beachten ist jedoch, dass mit steigender Länge der Subtrellisse zugleich der Parallelisierungsgewinn sinkt. Die direkte Methode stellt daher einen Kompromiss zwischen Leistungsfähigkeit und Implementierungsaufwand dar.

Überlappungsmethode Eine alternative Methode die Zustandswahrscheinlichkeiten an den Grenzen eines Subtrellis zu bestimmen,

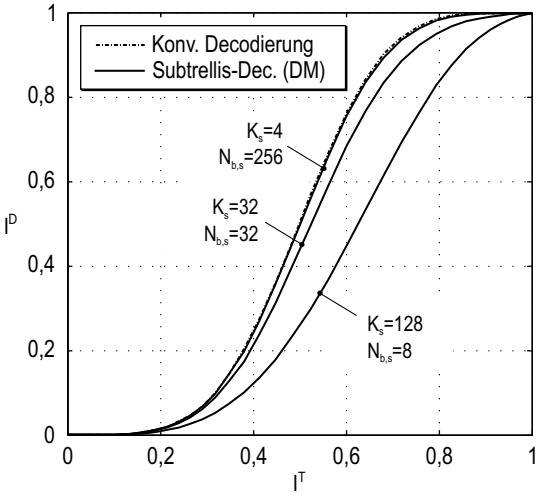


Abb. 6.7: EXIT-Diagramm einer Subtrellis-Decodierung mit direkter Methode (DM) (SCC mit Faltungscode $\mathbf{C} = [5, 7]_8$)

ist diese zu *schätzen* [MBJ06]. Der zu decodierende Subtrellis der Länge $N_{b,s}$ wird dazu um $N_{b,o}$ Trellis-Übergänge in beide Richtungen erweitert. In den entstehenden Überlappungsbereichen wird lediglich die Forward- bzw. Backward-Recursion durchgeführt, *ohne* die eigentlichen LLRs der Code- bzw. Infobits zu bestimmen. Die Subtrellis-Decodierung findet nach wie vor nur für die Teilsequenz der Länge $N_{c,s}$ statt. Abbildung 6.8 zeigt schematisch die Überlappungsmethode für einen Subtrellis.

Werden die Zustandswahrscheinlichkeiten an den Grenzen der künstlich verlängerten Subtrellisse als gleichwahrscheinlich angenommen, wird durch die Forward- und Backward-Recursion im Überlappungsbereich die Zuverlässigkeit der Zustandswahrscheinlichkeiten an den eigentlichen Grenzen der Subtrellisse erhöht. Für den Spezialfall $N_{b,o} = 0$ entspricht dieses Vorgehen der direkten Methode.

Die Überlappungsmethode wurde ebenso für den Faltungscode $\mathbf{C} = [5, 7]_8$ und eine LLR-Sequenz $L^{\mathrm{i}}(\mathbf{c})$ der Länge $N_c = 2048$ evaluiert. Da die Leistungsfähigkeit der direkten Methode besonders für $K_s = 128$

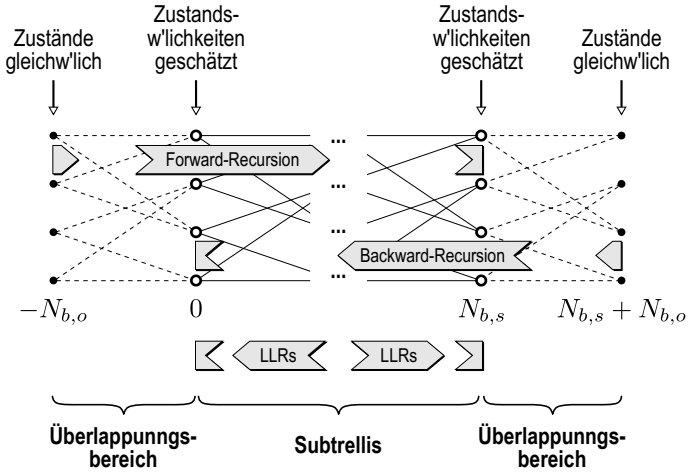


Abb. 6.8: Schematische Darstellung der Subtrellis-Decodierung mit Überlappungsmethode

und $K_s = 32$ Teilsequenzen gering ist, werden diese Fälle für Überlappungen von $N_{b,o} = 1$, 2, 4 und 8 erneut untersucht.

Das EXIT-Diagramm in Abbildung 6.9 verdeutlicht, dass mit steigender Überlappung die Zuverlässigkeit der Zustandswahrscheinlichkeiten an den Grenzen eines Subtrellis zunimmt und sich dadurch die gesamte Max*-Log-MAP Decodierung verbessert. Bereits für $N_{b,o} = 8$ ist in beiden Fällen mit einem nahezu perfekten Verlauf zu rechnen. Vergleichbare Ergebnisse wurden ebenso für diverse andere Faltungscodes in den Simulationen beobachtet.

Zusammenfassend stellt das Überlappen eine effiziente Methode zur Subtrellis-Decodierung dar. Selbst bei einem hohen Parallelisierungsgewinn durch viele Subdecoder kann die Leistungsfähigkeit der gesamten Decodierung durch entsprechende Wahl der Überlappungslänge angepasst werden. Der zusätzliche Implementierungsaufwand ist dabei überschaubar, da in den Überlappungsbereichen lediglich die Forward- bzw. Backward-Recursion durchgeführt wird und die dazu benötigen Übergangswahrscheinlichkeit schon vorliegen.

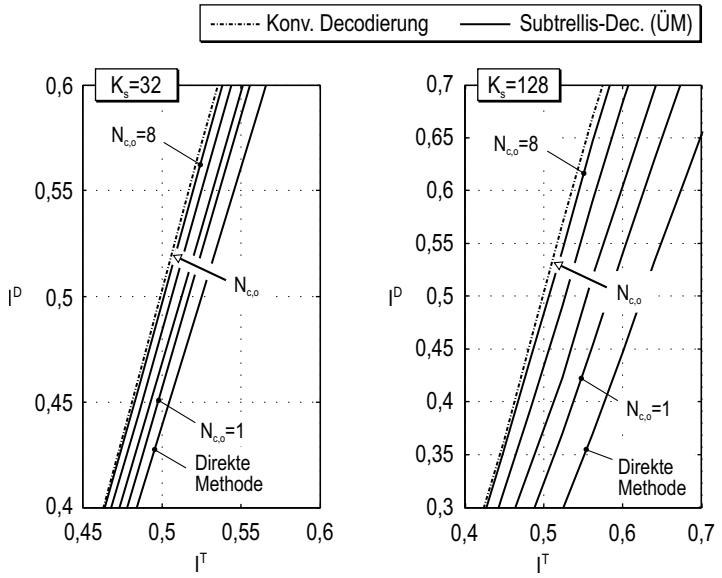


Abb. 6.9: EXIT-Diagramm einer Subtrellis-Decodierung mit Überlappungsmethode (ÜM) für $N_{b,o} = \in \{1, 2, 4, 8\}$ (SCC mit Faltungscode $\mathbf{C} = [5, 7]_8$)

6.3.4 GPP-Implementierung

Die Evaluation des Parallelisierungsgewinns einer Subtrellis-Decodierung wird auf einem Intel Core i7 GPP [Ii714] mit vier Prozessorkernen durchgeführt. Die Festkommaimplementierung des Max*-Log-MAP Decoders aus Abschnitt 5.4.4 wird dazu entsprechend modifiziert und die Rechenzeit für die Decodierung einer LLR-Sequenz $L^{\mathrm{i}}(\mathbf{c})$ der Länge $N_c = 2048$ für einen SCC mit $\mathbf{C} = [5, 7]_8$ gemessen.

In Anbetracht dessen, dass konventionelle GPPs eine eher geringe Anzahl an Prozessorkernen (2, 4, 8) besitzen, eignet sich speziell in diesem Fall die direkte Methode zur Bestimmung der Zustandswahrscheinlichkeiten an den Grenzen eines Subtrellis. Für den hier betrachteten GPP mit 4 Kernen bietet sich die Zerlegung in $K_s = 4$ Teilsequenzen mit je einer Subtrellis-Länge von $N_{b,s} = 256$ an. Gemäß Abbildung 6.7 ist die Leistungsfähigkeit dieser Parametrisierung nahezu mit der einer

nicht-parallelisierten Max*-Log-MAP Decodierung identisch. Darüber hinaus wird der Parallelisierungsgewinn für $K_s = 2$ und 8 untersucht.

Die Implementierung wird in C vorgenommen und mit dem Compiler der GNU Compiler Collection (Version 4.8.1) auf einem Linux Rechner (Ubuntu 13.10) kompiliert. Als Parallelisierungsschnittstelle wird POSIX Threads [Pth95] verwendet. Die prinzipielle Umsetzung zeigt Listing 6.3. Jeder Subdecoder wird als Thread instanziiert und anschließend zur Laufzeit durch das Betriebssystem auf einen Prozessorkern abgebildet.

Listing 6.3: Subtrellis-Decodierung mit direkter Methode (POSIX Thread Code-Schnipsel)

```
1  /* Erzeuge alle Subdecoder */
2  for (k=0; k<K; ++k) {
3
4      /* Uebergebe Teilsequenz an Subdecoder k */
5      subdecoder_data[k].LLR = LLR_i[k*N_b_s];
6
7      /* Erzeuge Subdecoder k als Thread */
8      pthread_create(&subdecoder[k], &attribute,
9                     subdecoder_fcn,
10                    (void*)&subdecoder_data[k]);
11 }
12
13 /* Warte bis alle Subdecoder fertig sind */
14 for (k=0; k<K; ++k)
15     pthread_join(subdecoder[k], NULL);
```

Die Ergebnisse dieser Implementierung sind in Tabelle 6.4 dargestellt und zeigen, dass durch die Subtrellis-Decodierung für die hier betrachtete Systemkonfiguration eine Beschleunigung von maximal $B_4 = 3{,}33$ bei einer Effizienz von $E_4 = 0{,}83$ möglich ist. Die Beschleunigung wird dabei nicht durch die Anzahl an Subdecodern K_s, sondern durch die Anzahl an verfügbaren parallelen Verarbeitungseinheiten K bestimmt. Erneut entsteht durch die Parallelisierungsschnittstelle ein Overhead, der durch den einfachen Faltungscode nicht vollständig kompensiert werden kann. Demnach ist für komplexere Codes mit einer höheren Effizienz zu rechnen.

Tab. 6.4: Ergebnisse der Parallelisierung durch Subtrellis-Decodierung mit direkter Methode

K_s	**Infobitrate** (Mbit/s)	**Beschleunigung**$^{\spadesuit}$
2	41,67	1,7
4	83,33	3,3
8	83,33	3,3
$\spadesuit$: Im Vergleich zur konv. Impl. mit 25 Mbit/s (s. Tab. 5.7)		

6.3.5 GPU-Implementierung

Die Vielzahl an Nebenläufigkeiten, die in den vorherigen Abschnitten vorgestellt wurden, werden im Folgenden durch eine Implementierung des Max*-Log-MAP Decoders auf einer GPU evaluiert. Das mit General Purpose Computation on GPUs (GPGPU) bezeichnete Vorhaben zielt darauf ab, die massive Parallelität eines solchen Vielkernprozessors (s. Abs. 2.1.1) für die Decodierung auszunutzen. Durch die Architektur eines Rechners vorgegeben, fungiert die GPU dabei als eine Art Co-Prozessor zum GPP. Ähnliche Implementierung von Turbo-Decodern auf GPUs finden sich in [WSC10] und [YC12].

Zur Implementierung des Decoders auf der GPU wird OpenCL (Open Computing Language) [OCL14] verwendet. OpenCL ist eine Plattform- und Hersteller-unabhängige Schnittstelle zur Programmierung heterogener Multiprozessorsysteme und unterscheidet dazu zwischen *Host*, *Device* und *Framework* [MGM+11]. Während der Host, hier der GPP, zur Steuerung und für den Datenaustausch mit dem Device, hier die GPU, zuständig ist, handelt es sich beim Framework um die Hersteller-spezifische Umsetzung der OpenCL-Schnittstelle für eine bestimmte GPU (z.B. Nvidia CUDA [NCU14]). Ein Device wird in mehrere Computing Units unterteilt, die wiederum die Processing Elements, die eigentlichen Prozessorkerne der GPU, enthalten. Eine Nvidia GeForce GTX 760 GPU [NGG14] besitzt z.B. 1152 Prozessorkerne, die sich auf 6 Computing Units mit je 192 Processing Elements aufteilen. Das Modell in Abbildung 6.10 verdeutlicht die OpenCL-Architektur.

Die eigentliche Signalverarbeitung wird mittels OpenCL Kernels aus-

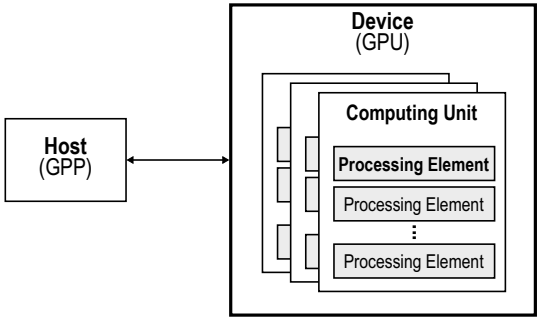


Abb. 6.10: Modell der OpenCL-Architektur

geführt. Ein Kernel ist die Umsetzung einer Operation, die parallel ausgeführt werden soll. Zur Erläuterung zeigt das Beispiel in Listing 6.4 den Kernel einer elementweisen Multiplikation zweier Vektoren `a` und `b`. Die Funktion `vec_mul` stellt dabei eine Art Prototyp dar, die das Ergebnis `c[k]` nur für einen spezifischen Index `k` berechnet. Zur Durchführung einer Vektormultiplikation der Länge N wird mittels OpenCL ein Indexraum gleicher Länge aufgespannt und für jeden Index eine Instanz des Kernels ausgeführt. Jeder Instanz wird dazu ein eindeutiger Identifikator (*engl.* Identification, ID) zugeteilt, der es ermöglicht, den entsprechenden Index abzuleiten. Die Instanzen können anschließend *parallel* auf den zur Verfügung stehenden Processing Elements abgearbeitet werden. Bei konventionellen GPUs mit >1000 Processing Elements hat dies eine massive Parallelisierung zur Folge.

Listing 6.4: Vektormultiplikation (OpenCL Code-Schnipsel)

```
1  __kernel void vec_mul(__global const float *a,
2                        __global const float *b,
3                        __global float *c,
4  {
5    /* ID auslesen und Index k ableiten */
6    int k = get_global_id(0);
7
8    /* Multiplikation zweier Elemente mit Index k */
9    c[k] = a[k] * b[k];
10 }
```

Zur Implementierung des Max*-Log-MAP Decoders werden die vorgestellten Parallelisierungstechniken wie folgt kombiniert: Die Berechnung der Zustandswahrscheinlichkeiten wird gemäß Abschnitt 6.3.1 als Kernel realisiert und somit vollständig parallelisiert. Des Weiteren werden die Subtrellis-Decodierung aus Abschnitt 6.3.3 mit der Überlappungsmethode angewendet. Die Berechnung der LLRs am Ausgang erfolgte dabei parallel zur Forward- und Backward-Recursion nach Überschreiten der Subtrellis-Mitte (s. Abs. 6.3.2).

Bei der Implementierung gilt es zu beachten, dass der aufgespannte Indexraum zur Instanziierung und Ausführung der Kernels an die jeweilige Architektur der GPU angepasst werden muss. OpenCL bietet dazu die Möglichkeit, die Instanzen gezielt auf unterschiedliche Computing Units zu verteilen oder mehrdimensionale Indexräume zu erzeugen. Des Weiteren wird die Speichernutzung der Implementierung analysiert. Während der Host und das Device über den globalen Speicher im Device kommunizieren, verfügt das Device zusätzlich über private und lokale Speicher, auf die nur von Processing Elements bzw. Computing Units zugegriffen werden kann. Da die Zugriffsgeschwindigkeit mit der Nähe zu einem Processing Element steigt (global→lokal→privat), wird die Speichernutzung dahingehend optimiert. Aus diesem Grund wird auch auf eine Festkommaimplementierung mit speicherintensiven Lookup-Tables verzichtet und statt dessen eine Gleitkommadarstellung verwendet.

Die Ergebnisse der OpenCL-Implementierung auf einer Nvidia GeForce GTX 760 GPU [NGG14] sind in Tabelle 6.5 gezeigt. Die zu decodierende LLR-Sequenz $L^{\mathrm{i}}(\mathbf{c})$ besitzt eine Länge von $N_c = 2048$ und wird durch einen rekursiven SCC mit $\mathbf{C} = [1, 15; 17]_8$ encodiert. Der Parallelisierungsgewinn wird für $K = 32$ und 64 Teilsequenzen sowie Überlappung von $N_{b,o} = 4$, 8 und 16 evaluiert.

Mit Infobitraten von bis zu 62,33 Mbit/s zeigt sich die enorme Rechenkapazität der GPU. Durch die massive Parallelisierung des Decoders auf unterschiedlichen Ebenen ist es möglich, die zur Verfügung stehenden 1152 Prozessorkerne optimal auszunutzen und eine hohe Effizienz der Implementierung zu erreichen. Die prinzipiell höheren Raten für eine Umsetzung mit $K = 32$ Subdecodern sind dadurch zu erklären, dass in diesem Fall die verschiedenen Speicher effizienter ausgenutzt

Tab. 6.5: Ergebnisse der Parallelisierung durch Subtrellis-Decodierung mit Überlappungsmethode

K_s	$N_{b,o}$	**Infobitrate** (Mbit/s)
32	4	62,33
	8	58,91
	16	55,80
64	4	58,82
	8	53,92
	16	47,75

und unnötige Lese- und Schreibzugriffe in langsame Speicherbereiche vermieden werden. Des Weiteren ist zu erkennen, dass infolge größerer Überlappungsbereiche und dem damit steigenden Signalverarbeitungsaufwand die Infobitrate erwartungsgemäß sinkt.

6.4 SC-MMSE-FD Turbo-Entzerrer

Der in diesem Abschnitt verfolgte Ansatz findet aus Sicht der Signalverarbeitung auf der obersten Ebene statt und zielt darauf ab, den gesamten SC-MMSE-FD Turbo-Entzerrer zu parallelisieren. Aufgrund des iterativen Entzerrens und Decodierens einer Empfangssequenz stellt die Turbo-Entzerrung ein vermeintlich nicht-parallelisierbares Verfahren dar. Eine SISO-Komponente verarbeitet jeweils Informationen, die zuvor von der anderen Komponente erzeugt wurden. Diese Datenabhängigkeit erfordert eine abwechselnde Ausführung des Entzerrers und Decoders. Nichtsdestotrotz kann diese Abhängigkeit durch eine zusätzliche Latenz mithilfe einer Software-Pipeline vollständig umgangen werden. Die dadurch erzeugte Unabhängigkeit bietet erneut Potential zur Parallelisierung und wird im Folgenden vorgestellt.

6.4.1 Pipelining

Software-Pipelining (*kurz* Pipelining) ist eine Form der Parallelisierung und wird in der Informatik zur Steigerung des Durchsatzes von Prozessen angewendet [Isa09]. Ein Prozess wird dazu in eine Serie (Pipeline) von Subprozessen zerlegt, bei der jeder Subprozess (Pipe) die Ausgangsdaten des vorherigen verarbeitet. Der Datenaustausch wird dabei durch zusätzliche Speicherelemente realisiert. Der Vorteil des Pipelinings liegt in der erzeugten Unabhängigkeit der einzelnen Pipes und der somit möglichen Parallelisierung.

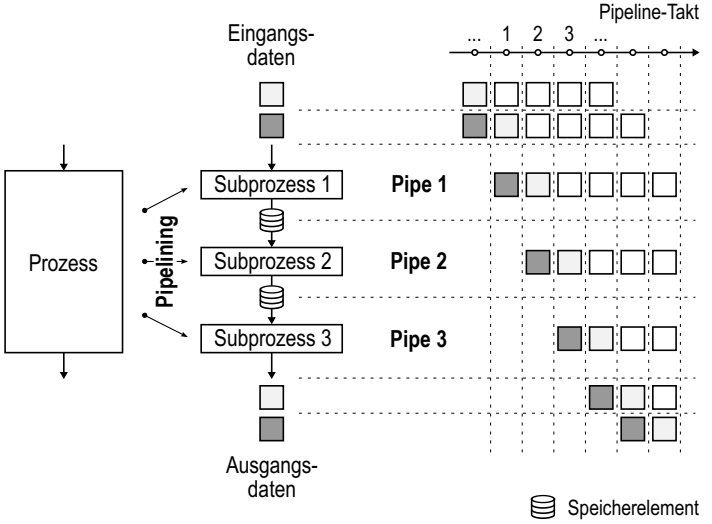


Abb. 6.11: Dreistufige Pipeline

Den prinzipiellen Ablauf des Pipelinings zeigt Abbildung 6.11 exemplarisch für drei Subprozesse. Zu Beginn sind die Speicher der Pipeline noch nicht mit Daten gefüllt. Während des ersten Pipeline-Takts werden die anstehenden Eingangsdaten von Subprozess 1 geladen, verarbeitet und anschließend in den ersten Speicher geschrieben. Im zweiten Pipeline-Takt verarbeitet Subprozess 2 die Daten im ersten Speicher während Subprozess 1 *zeitgleich* die nächsten Eingangsdaten verarbeitet und den Inhalt des Speichers überschreibt. Zum Ende des zweiten

Pipeline-Takts ist nun auch der zweite Speicher mit Daten aus Subprozess 2 gefüllt. Zu Beginn des dritten Pipeline-Takts ist die Pipeline vollständig gefüllt und Subprozess 3 verarbeitet die Daten im zweiten Speicher. Erste Ausgangsdaten sind in diesem Beispiel somit zum Ende des dritten Pipeline-Takts vorhanden. Im Weiteren Verlauf des Pipelinings werden mit jedem Pipeline-Takt neue Ausgangsdaten erzeugt.

Der Vorteil dieser Fließband-ähnlichen Verarbeitung liegt wie bereits erwähnt in der Unabhängigkeit der Pipes. Diese können auf parallele Verarbeitungseinheiten abgebildet und *zeitgleich* ausgeführt werden. Bei ausgeglichener Auslastung der einzelnen Subprozesse (*engl.* Balanced Configuration) erhöht sich somit der Durchsatz eines Prozesses um die Anzahl an Pipes. Nichtsdestotrotz wird ebenso die Latenzzeit der Verarbeitung erhöht. Von der Ein- bis zur Ausgabe der Daten vergehen beispielsweise im obigen Beispiel drei Pipeline-Takte. Zusätzlich muss ausreichend Speicherplatz zum temporären Vorhalten der Daten bereitstehen.

Anwendung auf die Turbo-Entzerrung Aus Sicht des Pipelinings stellt die SC-MMSE-FD Turbo-Entzerrung einen gewöhnlichen Prozess dar, bei dem abwechselnd eine SC-MMSE-FD Entzerrung und Max*-Log-MAP Decodierung durchgeführt werden. Dieser Prozess kann gemäß des Pipeline-Prinzips gleichermaßen in Subprozesse zerlegt und parallelisiert werden [STJ13]. Der kleinst mögliche Subprozess ist dabei *eine* SISO-Komponente. Durch Gruppieren mehrerer *Entzerr-* und *Decodierprozesse* lässt sich der Umfang eines Subprozesses erhöhen. Da sich die Rechenkomplexität abhängig von der Komplexität des Faltungscodes sehr stark zwischen SC-MMSE-FD Entzerrer und Max*-Log-MAP Decoder unterscheiden kann, sollte mit dem Ziel einer Balanced Configuration ein Subprozess aus einer *geraden* Anzahl an SISO-Komponenten bestehen.

Zur Umsetzung der Pipeline müssen zusätzlich das (De-)Interleaving und Soft-(De-)Mapping berücksichtigt werden. Da diese Vorgänge im Vergleich zu den SISO-Komponenten jedoch einen geringen Signalverarbeitungsaufwand verursachen, wurde das Interleaving und Soft-

Mapping dem Decodierprozess und das Soft-Demapping und Deinterleaving dem Entzerrprozess zugeordnet.

Abbildung 6.12 zeigt die Umsetzung der SC-MMSE-FD Turbo-Entzerrung durch eine dreistufige Pipeline, bei der eine Pipe eine Turbo-Iteration realisiert. Zu Beginn sind die Speicher der Pipeline leer und die Beobachtung $\mathbf{r}$ wird im ersten Pipeline-Takt in Pipe 1 entzerrt und decodiert. Während des zweiten Pipeline-Takts verarbeitet Pipe 1 bereits eine neue Beobachtung, während Pipe 2 die erneute Entzerrung und Decodierung der vorherigen Beobachtung mithilfe der Symbole $\bar{\mathbf{s}}$ im Speicher durchführt. Nach drei Pipeline-Takten ist die Turbo-Entzerrer-Pipeline gefüllt. Im weiteren Verlauf werden in jedem Pipeline-Takt drei Iterationen *parallel* ausgeführt. Die decodierte Infobit-Sequenz $\hat{\mathbf{b}}$ des Max*-Log-MAP Decoders in der letzten Pipe wird an die folgende Signalverarbeitung weitergegeben. Die anderen Zwischenergebnisse können als Grundlage für ein Abbruchkriterium dienen.

Analyse des Pipelinings Die folgende Untersuchung des *theoretischen* Parallelisierungsgewinns und Speicherbedarfs bezieht sich auf die Implementierung eines SC-MMSE-FD Turbo-Entzerrers, bei dem eine Pipe einer Turbo-Iteration entspricht und die Pipeline aus $1 + N_{\mathrm{it}}$ Stufen besteht.

Zunächst kann die Beschleunigung durch das Pipelining ermittelt werden. Davon ausgehend, dass K unabhängige Verarbeitungseinheiten zur Verfügung stehen und die Verarbeitungsdauer der Pipes gleich groß ist, gilt für die Beschleunigung

$$B_K = \frac{1 + N_{\mathrm{it}}}{\left\lceil \frac{1+N_{\mathrm{it}}}{K} \right\rceil} \,. \tag{6.4}$$

Zur Erläuterung dieses Zusammenhangs wird ein Turbo-Entzerrer mit $N_{\mathrm{it}} = 9$ Iterationen (10 Pipes) und vier zur Verfügung stehende Prozessoren ($K = 4$) betrachtet. Die Verarbeitungsdauer pro Pipe beträgt T. Da nicht alle Pipes direkt parallel durch die Prozessoren verarbeitet werden können, wird eine Zeit von $T_4 = \left\lceil \frac{1+N_{\mathrm{it}}}{K} \right\rceil T = 3T$ benötigt, bis alle Pipes abgearbeitet sind. Die Beschleunigung gegenüber

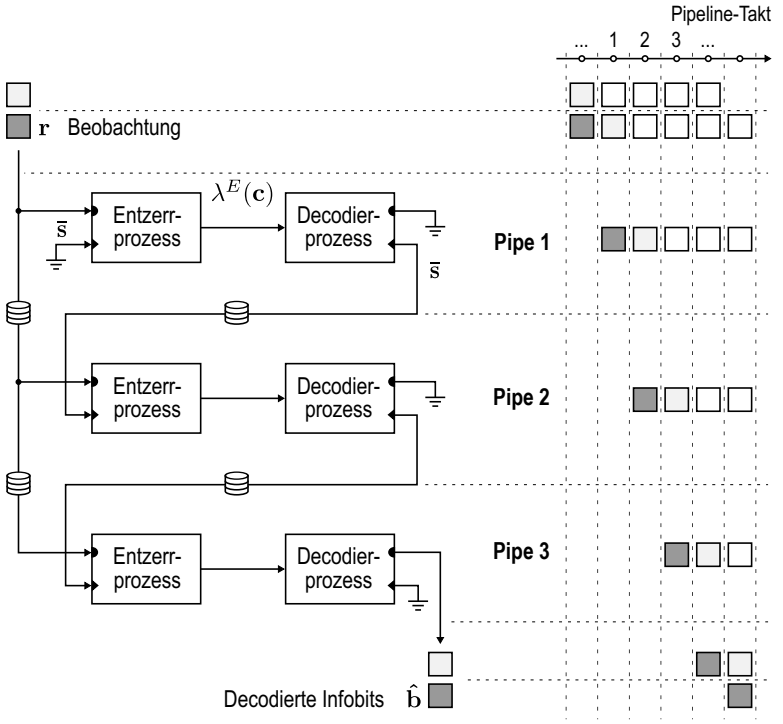


Abb. 6.12: Dreistufige Turbo-Entzerrer-Pipeline

einer konventionellen Implementierung ohne Pipelining mit einer Verarbeitungsdauer von $T_1 = (1 + N_{\mathrm{it}})T = 10T$, entspricht gemäß (2.1) $B_4 = T_1/T_4 = 3{,}3$.

Der zusätzliche Speicherbedarf wird durch die Speicherung der Pipe-Ausgänge und das Verzögern der Beobachtung verursacht. Insgesamt müssen $2N_{\mathrm{it}}$ Sequenzen mit je N_s komplexwertigen Symbolen temporär im Speicher abgelegt werden. Für das Beispiel in Abbildung 6.12 sind es demnach $4N_s$ komplexe Zahlenwerte.

Des Weiteren wird durch das Pipelining eine zusätzliche Latenz in der Signalverarbeitung erzeugt. Bis die decodierte Infobit-Sequenz einer Beobachtung $\mathbf{r}$ vorliegt, vergehen $1 + N_{\mathrm{it}}$ Pipeline-Takte. Wird ange-

nommen, dass die Symbolrate 10 MHz und die Blocklänge $N_s = 1024$ beträgt, folgt bei einer echtzeitfähigen Verarbeitung für das in Abbildung 6.12 gezeigte Beispiel eine Latenz von rund 0,3 ms. Verglichen mit den Qualitätsanforderungen zeitkritischer Dienste, wie z.B. VoIP (Voice over Internet Protocol), ist diese Dauer jedoch gering[1], sodass die verursachte Latenz zugunsten des Parallelisierungsgewinns toleriert werden kann. Im Einzelfall gilt es für konkrete Systemkonfigurationen eine Verzögerungsbilanz des Gesamtsystems aufzustellen.

6.4.2 GPP-Implementierung

Die Evaluation der Leistungsfähigkeit einer Pipeline-Implementierung erfolgt auf einem Intel Core i7 GPP [Ii714] mit vier Prozessorkernen. Hierzu wird die SIMD-Implementierung des SC-MMSE-FD Entzerrers aus Abschnitt 6.2.4 und die Festkommaimplementierung des Max*-Log-MAP Decoders aus Abschnitt 5.4.4 zur Umsetzung des SC-MMSE-FD Turbo-Entzerrers herangezogen. Die Systemkonfiguration für diese Implementierung lautete wie folgt:

- QPSK-modulierte Symbole
- $N_s \in \{512; 1024; 2048; 4096\}$
- SCC mit Faltungscode $[5, 7]_8$
- Zufallsinterleaver
- $N_{\mathrm{it}} \in \{0; 1; 3; 8\}$

Als Parallelisierungsschnittstelle wird wie bereits in Abschnitt 6.2.4 OpenMP verwendet. Die prinzipielle Umsetzung als C-Code zeigt Listing 6.5. Die einzelnen Pipes entsprechen erneut je einer Turbo-Iteration und werden in einer Schleife abgearbeitet. Durch die Steueranweisung `parallel for` wird jeder Durchlauf bzw. jede Pipe als Thread ausgeführt. Das Betriebssystem übernimmt anschließend zur Laufzeit

[1] Laut ITU-T [Int94] wird für hoch-interaktive Kommunikation eine maximale Latenz von 150 ms für Sprachkommunikation empfohlen.

die Abbildung der Threads auf die Prozessorkerne. Neben der Parallelisierung ist zusätzlich die Implementierung eines Speichermanagements vorzunehmen, das die Inhalte nach jeder Pipe mit den neuen Ergebnissen überschreibt. Mithilfe zirkulärer Buffer und C-Pointer-Arithmetik lässt sich dies effizient realisieren.

Listing 6.5: Pipelining der Turbo-Entzerrung (OpenMP Code-Schnipsel)

```
1 /* Initalisierung */
2 r[0]      = r;
3 s_bar[0] = zero_vector;
4
5 /* Pipeline */
6 #pragma omp parallel for
7 for (int it=0; it<=Nit; it++) {
8
9   /* Pipe (Verwende zirkulaere Buffer) */
10  Equalize_Process(lambda_E,r[it],s_bar[it]);
11  Decode_Process(b_hat[it],s_bar[it+1],lambda_E);
12
13  /* Schreibe in zirkulaeren Buffer */
14  r[it+1]=r[it];
15 }
```

Ergebnisse Tabelle 6.6 zeigt die gemessen Symbolraten für die eine echtzeitfähige SC-MMSE-FD Turbo-Entzerrung unter der gegebenen Systemkonfiguration noch möglich ist. Zur Verifizierung der Absolutwerte werde exemplarisch die Sequenzlänge $N_s = 1024$ und $N_{\text{it}} = 0$ für eine sequentielle Implementierung betrachtet. Die Symboldauer von 57,5 ns (1/17,39 MHz) setzt sich aus den 40 ns zur Decodierung (1/25 Mbit/s, s. Tab. 5.7) und den 9,8 ns zur Entzerrung (1/102,04 MHz, s. Tab. 6.2) zusammen. Die restliche Dauer von 7,7 ns wird zum Soft-(De-)Mapping und (De-)Interleaving benötigt.

Bei der Auswertung der Beschleunigung wird erkennbar, dass durch die Parallelisierung erneut ein konstanter Aufwand für den Kontextwechsel entsteht (s. Abs. 6.2.4). Dieser Overhead macht sich besonders für $N_{\text{it}} = 0$ negativ bemerkbar, da in diesem Fall die sequentielle Implementierung schneller als die einstufige Pipeline ist. Diese Situation

Tab. 6.6: Ergebnisse der Parallelisierung durch Pipelining

N_s	N_{it}	**Symbolrate** (MHz)		**Beschleunigung**
		Sequentiell	**Pipelining**	
512	0	17,24	15,63	0,91
	1	8,58	14,71	1,71
	3	4,30	13,70	3,18
	8	1,91	5,20	2,72
1024	0	17,39	15,87	0,91
	1	8,64	15,04	1,74
	3	4,30	14,29	3,32
	8	1,92	5,24	2,73
2048	0	17,12	15,65	0,91
	1	8,58	15,06	1,76
	3	4,28	14,68	3,43
	8	1,90	5,19	2,73
4096	0	16,67	15,24	0,91
	1	8,35	14,10	1,69
	3	4,17	13,46	3,23
	8	1,86	4,97	2,67

kann jedoch bereits bei der Umsetzung im C-Code abgefangen werden, indem die Parallelisierung deaktiviert wird.

Für alle anderen Fälle $N_{\text{it}} > 0$ kann eine eindeutige Beschleunigung durch das Pipelining erreicht werden. So beträgt z.B. der Geschwindigkeitszuwachs für vier Pipes ($N_{\text{it}} = 3$) maximal $B_4 = 3{,}43$ bei einer Effizienz von $E_4 = 0{,}86$. Die geringere Beschleunigung für den Fall von neun Pipes ($N_{\text{it}} = 8$) ergibt sich aus der Tatsache, dass in dieser Konstellation alle Prozessorkerne zwei Pipes durchführen und anschließend auf einen Kern warten, der die verbleibende neunte Pipe abarbeitet.

Die Beschleunigung durch Pipelining wird darüber hinaus gesteigert, wenn die Problemgröße pro Pipe den Aufwand des Kontextwechsels dominiert. Da die Komplexität der Max*-Log-MAP Decodierung exponentiell von der Länge des Schieberegisters im Encoder abhängt, tritt diese Situation ein, wenn ein aufwändigerer Fehlerschutz verwendet wird. Für einen Faltungscode gemäß des IEEE 802.11 Standards

($\mathbf{C} = [133, 171]_8$) folgt z.B. für vier Pipes und $N_s = 1024$ ein Beschleunigung von $B_4 = 3{,}68$ bei einer Effizienz von $E_4 = 0{,}92$ [SJ13].

6.5 SCCC-Decoder

Wie bereits in Abschnitt 4.3.5 beschrieben, ist ein SCCC-Decoder aus zwei gekoppelten Max*-Log-MAP Decodern aufgebaut, die über einen Interleaver bzw. Deinterleaver miteinander verbunden sind. Die Ansätze aus Abschnitt 6.3 lassen sich deshalb auch zur Parallelisierung jedes *einzelnen* Decoders anwenden.

Ebenso ist es möglich, den SCCC-Decoder als Pipeline zu implementieren. Da in einem sequentiellen Prozess abwechselnd der innere und äußere Max*-Log-MAP Decoder ausgeführt wird, lassen sich Subprozesse erzeugen. Der kleinst mögliche Subprozess ist dabei ein SISO-Decoder, wobei das Deinterleaving dem inneren und das Interleaving dem äußeren Decoder zugeordnet wird. Da es sich im Vergleich zum äußeren Code beim inneren um einen simplen, rekursiven Rate-1 Code handelt, sollte die Größe eines Subprozesses für eine Balanced Configuration einer geraden Anzahl von Decodern entsprechen. Die Zusammenfassung beider Decoder zu einer Pipe entspricht analog zum SC-MMSE-FD Turbo-Entzerrer einer Decoder-Iteration als Subprozess.

Die Analysen des Pipelinings aus Abschnitt 6.4 sind prinzipiell auf den SCCC-Decoder übertragbar. Gemäß dem Fall, dass eine Pipe einer Decoder-Iteration entspricht, folgt mit (6.4) für die theoretische Beschleunigung B_K bei K unabhängigen Verarbeitungseinheiten

$$B_K = \frac{1 + N_{\text{it}}^D}{\left\lceil \frac{1+N_{\text{it}}^D}{K} \right\rceil} \,. \tag{6.5}$$

Zusätzlicher Speicher wird zum Vorhalten von $2N_{\text{it}}^D$ LLR-Sequenzen der Länge N_c benötigt. Die Latenz der Verarbeitung wird erneut durch das Pipelining erhöht. Bis die decodierte Infobit-Sequenz einer LLR-Sequenz $\lambda^E(\mathbf{c})$ vorliegt, vergehen $1 + N_{\text{it}}^D$ Pipeline-Takte.

6.6 Zusammenfassung

Die Untersuchungen in diesem Kapitel zeigen, dass sich eine Parallelisierung der SC-MMSE-FD Turbo-Entzerrung auf unterschiedlichen Ebenen vornehmen lässt und dadurch die Effizienz einer Implementierung deutlich gesteigert werden kann. Dazu wurden Nebenläufigkeiten innerhalb der SISO-Komponenten und für das gesamte Turbo-System aufgezeigt und analysiert. Der Umfang der Untersuchungen reichte von elementarsten arithmetischen Operationen auf Datenebene bis hin zur Iterationsebene des Verfahrens. Die Parallelisierungen wurden mithilfe von Implementierungen evaluiert und der Gewinn ausgewertet.

Im Fall der SC-MMSE-FD Entzerrung erweist sich die Parallelisierung der elementweisen Operationen als äußert wirksam. Diese wird durch die Frequenzbereichsentzerrung ermöglicht und erzielt durch SIMD-Befehlssätze eine rund 6-fache Beschleunigung der Entzerrung bei einer Symbolrate von rund 100 MHz auf einem GPP. Der Parallelisierungsgewinn wird dabei durch die Integration einer hochoptimierten FFT-Bibliothek verstärkt.

Aus Sicht der Parallelisierung bilden die Datenabhängigkeiten der Forward- und Backward-Recursion den Flaschenhals des Max*-Log-MAP Decoders. Diese können jedoch durch das Verfahren der Subtrellis-Decodierung partiell aufgelöst werden. Die dadurch erzeugte Unabhängigkeit der Subdecoder ermöglicht schließlich eine parallele Implementierung der gesamten Decodierung. Zur Initialisierung der Subdecoder wurden zwei Methoden vorgestellt, um die Zustandswahrscheinlichkeiten an den Grenzen eines Subtrellis zu bestimmen. Mithilfe der Subtrellis-Decodierung und der Überlappungsmethode kann eine Infobitrate von rund 60 Mbit/s auf einer GPU, bei nahezu unveränderter Leistungsfähigkeit des Decoders, erreicht werden.

Neben der Parallelisierung der einzelnen SISO-Komponenten wurde die gesamte SC-MMSE-FD Turbo-Entzerrung hinsichtlich Nebenläufigkeiten untersucht. Hierbei konnte das Prinzip des Pipelinings auf das iterative Verfahren angewendet und so eine Parallelität auf Iterationsebene erzeugt werden. Der Ansatz wurde im Detail erläutert und

hinsichtlich des Parallelisierungsgewinns analysiert. Eine Implementierung auf einem Vierkern-GPP erreicht so bei vier Turbo-Iterationen eine Symbolrate von rund 14 MHz.

In Anbetracht aller Parallelisierungen ist die effektivste Implementierung einer SC-MMSE-FD Turbo-Entzerrung durch die *kombinierte* Rechenkapazität eines GPPs mit einer GPU gegeben. Während die GPU lediglich als Co-Prozessor zur Max*-Log-MAP Decodierung verwendet wird, übernimmt der GPP die SC-MMSE-FD Entzerrung sowie das (De-)Interleaving und Soft-(De-)Mapping. Dabei wird das Pipelining und die Parallelität auf Datenebene durch die SIMD-Architektur des GPPs ausgenutzt.

KAPITEL 7

Zusammenfassung und Ausblick

In dieser Arbeit wurde die Turbo-Entzerrung hinsichtlich ihrer Implementierungsaspekte auf einem Software Defined Radio untersucht. Als Kernkomponenten des iterativen Verfahrens wurde dabei ein SC-MMSE-FD Entzerrer und ein Max*-Log-MAP Decoder verwendet, die gegenüber dem klassischen MAP-basierten Ansatz einen Kompromiss zwischen Leistungsfähigkeit und Implementierungsaufwand ermöglichen. Als die zentralen Aspekte der Implementierung wurden die Umsetzung in Festkommaarithmetik sowie die Parallelisierbarkeit des Verfahrens behandelt.

Obwohl Festkommaimplementierungen in erster Linie für Logiken von Bedeutung sind, ermöglichen sie unabhängig davon eine effiziente Nutzung der verfügbaren Ressourcen. Voraussetzung dafür ist jedoch, dass die Festkommadarstellung an die entsprechende Signalverarbeitung angepasst wird. Die Analyse der Turbo-Entzerrung hat gezeigt, dass die Festkommadarstellung der Symbole und Parameter unterschiedlich stark von der Systemkonfiguration abhängt. Den größten Einfluss auf die Dimensionierung haben dabei das verwendete Modulationsverfahren, die Sequenzlängen und die Parameter des Fehlerschutzes. Die in dieser Arbeit hergeleiteten Modelle ermöglichen es, effiziente Festkommaimplementierungen zu entwerfen, ohne diese im Vorfeld vollständig

zu simulieren. Dies erspart einen hohen Simulationsaufwand und kann für beliebige Konfigurationen durchgeführt werden. Den Nutzen und die Richtigkeit der Modelle wurde durch Referenzimplementierungen verdeutlicht und verifiziert.

Parallelisierung ist und wird auch zukünftig eine wichtige Methode zur Optimierung darstellen. Vielkernprozessoren mit mehr als 1000 parallelen Verarbeitungseinheiten stellen den gegenwärtigen Stand der Technik dar. Zur effizienten Auslastung solcher Prozessoren müssen Implementierungen parallelisiert werden. Die Analysen hinsichtlich einer Parallelisierbarkeit der Turbo-Entzerrung haben verdeutlicht, dass das Verfahren diesbezüglich ein hohes Potential auf verschiedenen Ebenen der Signalverarbeitung bietet. Von der arithmetischen Ebene bis hin zur Iterationsebene wurden parallele Abläufe identifiziert und analysiert. Im Anschluss wurde der Parallelisierungsgewinn für verschiedene Prozessoren mithilfe von Referenzimplementierungen demonstriert. Die teils enormen Beschleunigungen belegen den Nutzen der Analysen und unterstreichen erneut den Stellenwert der Parallelisierung.

Ausblick

Aufgrund der heutigen Verbreitung von Mehrantennensystemen (*engl.* Multiple Input Multiple Output, MIMO) stellt die Erweiterung der Analysen auf diese Systeme eine logische Konsequenz dar. Da die Basis einer *MIMO-Turbo-Entzerrung* eine konventionelle Turbo-Entzerrung bildet, lässt sich nach einer ersten Einschätzung ein Großteil der Ergebnisse dieser Arbeit wiederverwenden.

Die Untersuchungen in dieser Arbeit setzten die vollständige Kanalzustandsinformation im Empfänger voraus. Fehler, die z.B. durch eine nicht-perfekte Kanalschätzung oder eine falsche Synchronisation entstehen werden, wurden nicht beachtet. Die Berücksichtigung dieser *Fehlereinflüsse* bei der Analyse der Festkommaaspekte stellt eine wichtige Voraussetzung für reale Funksysteme dar.

Zuletzt ist der Aufbau eines echtzeitfähigen *Demonstrators* zu nennen, der die Ergebnisse dieser Arbeit zur Implementierung einer vollständigen Funkübertragung auf einer heterogenen SDR-Plattform nutzt.

Abkürzungsverzeichnis

ALU	Arithmetic Logic Unit
ARM	Advanced RISC Machines
ASIC	Application-Specific Integrated Circuit
AVX	Advanced Vector Extensions
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BICM	Bit-Interleaved Coded Modulation
BPSK	Binary Phase Shift Keying
bzw.	beziehungsweise
CCDF	Complementary Cumulative Distribution Function
CDMA	Code Division Multiple Access
CLB	Configurable Logic Block
CP	Cyclic Prefix
CSIR	Channel State Information at the Receiver
CUDA	Compute Unified Device Architecture
D/A	Digital/Analog Wandlung
DFT	Diskreten Fourier-Transformation
DM	Direkte Methode
DSP	Digital Signal Processor
EXIT	Extrinsic Information Transfer
FD	Frequency Domain
FF	Flip-Flop
FFT	Fast Fourier Transform
FFTW	Fastest Fourier Transform in the West
FPGA	Field Programmable Gate Array
GNU	GNU is not UNIX
GPGPU	General-Purpose Computing on Graphics Processing Units

GPP	General Purpose Processor
GPU	Graphics Processing Unit
HF	Hochfrequenztechnik
ID	Identification
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
ISE	Integrated Software Environment
LLR	Log-Likelihood Ratio
LOS	Line of Sight
LSB	Least Significant Bit
LTE	Long Term Evolution
LUT	Lookup-Table
MAP	Maximum *a posteriori*
Mbit/s	Mega Bit per Second
MIMO	Multiple Input Multiple Output
MMSE	Minimum Mean Square Error
NLOS	Non-Line of Sight
OFDM	Orthogonal Frequency-Division Multiplexing
OpenCL	Open Compunting Language
OpenMP	Open Specifications for Multi Processing
POSIX	Portable Operating System Interface
PSK	Phase Shift Keying
PSS	Proportional Set Size
Pthreads	POSIX Threads
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase-Shift Keying
RAM	Random-Access Memory
RCPC	Rate-Compatible Punctured Convolutional
SC	Soft Interference Cancellation
SCC	Single Convolutional Code
SCCC	Serial Concatenated Convolutional Code
SDR	Software Defined Radio
SIMD	Single Instruction, Multiple Data
SISO	Soft-Input Soft-Output
SNR	Signal-to-Noise Ratio
SOVA	Soft Output Viterbi Algorithm
SR	Software Radio
SSE	Streaming SIMD Extensions

SVE	Signalverarbeitungseinheit
SVFuA	Streitkräftegemeinsame, verbundfähige Funkgeräteausstattung
TDL	Tapped-Delay-Line
ÜM	Überlappungsmethode
UMTS	Universal Mobile Telecommunications System
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral
usw.	und so weiter
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very-Large-Scale Integration
VoIP	Voice over Internet Protocol
VOLK	Vector-Optimized Library of Kernels
WLAN	Wireless Local Area Network
W'keit	Wahrscheinlichkeit
z.B.	zum Beispiel

Symbolverzeichnis

$(\cdot)^H$	Hermitesche Matrix oder Vektor
$(\cdot)^T$	Transponierte Matrix oder Vektor
$\lceil \cdot \rceil$	Ceiling-Operator
$[\cdot]_Q$	Quantisierter Wert
$*$	Faltungsoperator
$\alpha_k(\cdot)$	Wahrscheinlichkeit des Zustands (Forward-Recursion) mit Index k
$\underline{\alpha}_k(\cdot)$	Logarithmierte Wahrscheinlichkeit des Zustands (Forward-Recursion) mit Index k
$\beta_k(\cdot)$	Zustandswahrscheinlichkeit (Backward-Recursion) mit Index k
$\underline{\beta}_k(\cdot)$	Logarithmierte Zustandswahrscheinlichkeit (Backward-Recursion) mit Index k
b_k	Infobit mit Index k
$\mathbf{b}$	Infobit-Sequenz, $\in \{0,1\}^{N_b}$
$\hat{\mathbf{b}}$	Geschätzte Infobit-Sequenz, $\in \{0,1\}^{N_b}$
$\mathbf{B}$	Inverse Fourier-Matrix, $\in \mathbb{C}^{N_s \times N_s}$
B_0	Infobit während Trellis-Übergang
B_K	Beschleunigung auf K parallelen Verarbeitungseinheiten
$\gamma_k(\cdot,\cdot)$	Übergangswahrscheinlichkeit mit Index k
$\underline{\gamma}_k(\cdot,\cdot)$	Logarithmierte Übergangswahrscheinlichkeit mit Index k
$\underline{\gamma}_{\max}$	Maximum der logarithmierten Übergangswahrscheinlichkeit
Γ_{sat}	Sättingswert
Γ'_{sat}	Sättingswert nach Berechnung durch Tschebyscheff-Ungleichung
c_k	Codebit mit Index k
$\mathbf{c}$	Codebit-Sequenz, $\in \{0,1\}^{N_c}$
$\mathbf{C}$	Faltungscode (s. Gl. (3.10))

$\mathbf{C}_r$	Rekursiver Faltungscode des inneren Rate-1 Encoders im SCCC-Encoder
$C_{(\cdot)}$	Codebit während Trellis-Übergang
$\mathcal{C}$	Codierungsvorschrift
$\mathbb{C}$	Menge der komplexen Zahlen
$\mathcal{CN}$	Zirkulärsymmetrische Normalverteilung
δ	Höhe einer Amplitudenstufe bei der Quantisierung
d_k	Datenbit mit Index k
$\mathbf{d}$	Datenbit-Sequenz, $\in \{0,1\}^{N_c}$
D	Dynamikbereich
$\mathrm{E}\{\cdot\}$	Erwartungswert
E_b	Energie pro Bit
$E_b\sigma_0^2$	Signal-to-Noise Ratio (SNR) pro Bit
E_K	Effizient auf K parallelen Verarbeitungseinheiten
$\zeta(\cdot)$	*A priori* Information
$\zeta^E(c_k)$	*A priori* LLR am Eingang des SISO-Decoders
$\zeta^D(d_k)$	*A priori* LLR am Eingang des SISO-Entzerrers
f^E	Verlauf der Transinformation zw. d_k und $\lambda^E(d_k)$
f^D	Verlauf der Transinformation zw. c_k und $\lambda^D(c_k)$
$\mathbf{F}$	Fourier-Matrix, $\in \mathbb{C}^{N_s \times N_s}$
$\mathbf{F}_{k,l}$	Element der Fourier-Matrix in der k-ten Zeile und l-ten Spalte
$\mathbf{g}_{(\cdot)}$	Generatorpolynom (nicht-rekursiver SCC)
$\mathbf{g}_r$	Generatorpolynom (rekursiver SCC)
$h_{(\cdot)}(t)$	Kanalkoeffizient zum Zeitpunkt t
$h(t,\tau)$	Zeitvariante Kanalimpulsantwort (zeitkontinuierlich)
$\mathbf{h}$	Kanalimpulsantwort (zeitdiskret), $\in \mathbb{C}^P$
$\mathbf{h}(t)$	Zeitvariante Kanalimpulsantwort (zeitdiskret), $\in \mathbb{C}^P$
$\mathbf{H}$	Zyklischen Kanalfaltungsmatrix (s. Gl. (3.16)), $\in \mathbb{C}^{N_s \times N_s}$
$\mathbf{I}$	Einheitsmatrix, $\in \mathbb{C}^{N_s \times N_s}$
I^E	Transinformation zw. d_k und $\lambda^E(d_k)$
I^D	Transinformation zw. c_k und $\lambda^D(c_k)$
I^T	Transinformation zw. d_k und $\zeta^E(d_k)$ oder c_k und $\zeta^D(c_k)$
j	Imaginäre Element, $j = \sqrt{-1}$
k	Symbol oder Bit Index
K	Anzahl an parallelen Verarbeitungseinheiten
K_s	Anzahl an Subdecodern zur Subtrellis-Decodierung

$\lambda(\cdot)$	Extrinsische Information
$\lambda^E(d_k)$	Extrinsisches LLR am Ausgang des SISO-Entzerrers
$\lambda^D(c_k)$	Extrinsisches LLR am Ausgang des SISO-Decoders
Λ	Operator Soft-Mapping
Λ^{-1}	Operator Soft-Demapping
$\mathrm{ld}(\cdot)$	Logarithmus zur Basis 2 (Logarithmus Dualis)
$\ln(\cdot)$	Logarithmus zur Basis e (Natürlicher Logarithmus)
$\log(\cdot)$	Logarithmus zur Basis 10
$L(\cdot)$	Log-Likelihood Ratio
$L^i(c_k)$	Codebit-LLR am Eingang des Decoders
$L^i(b_k)$	Infobit-LLR am Eingang des Decoders
$L^o(c_k)$	Codebit-LLR am Ausgang des Decoders
$L^o(b_k)$	Infobit-LLR am Ausgang des Decoders
m	Anzahl an Integerbits
M	Stufigkeit des Modulationsverfahrens
ν	Länge des Schieberegisters im Encoder
n	Anzahl an Nachkommabits
n_b	Anzahl an encodierten Bits pro Takt
n_c	Anzahl an codierten Bits pro Takt
$n(t)$	Zeitkontinuierliches Rauschen
$\mathbf{n}$	Rauschen, $\in \mathbb{C}^{N_s}$
N	Wortbreite
N_b	Anzahl an Bits in einer Infobit-Sequenz
$N_{b,s}$	Länge eines Subtrellis
$N_{b,o}$	Länge des Überlappungsbereichs eines Subdecoders
N_c	Anzahl an Bits in einer Codebit-Sequenz
$N_{c,s}$	Länge einer Teilsequenz zur Subtrellis-Decodierung
N_{it}	Anzahl an Iterationen im Turbo-Entzerrer
N_{it}^D	Anzahl an Iterationen im SCCC-Decoder
N_s	Anzahl an Symbolen in einer Symbol-Sequenz
$\mathcal{N}$	Normalverteilung
$\mathcal{O}(\cdot)$	Landau-Notation, Bsp. $f \in \mathcal{O}(x) \rightarrow f$ wächst linear
$\mathbf{\Xi}$	Kanalübertragungsmatrix (s. Gl. (4.9)), $\in \mathbb{C}^{N_s \times N_s}$
π	Operator Interleaver
π^{-1}	Operator Deinterleaver
π_r	Permutationsschema Interleaver im SCCC
$p(\cdot)$	Wahrscheinlichkeitsdichte

P	Anzahl an Kanalkoeffizienten
P_{sat}	Sättigungswahrscheinlichkeit
$P(\cdot)$	Wahrscheinlichkeit
q	Quantisierungsfehler
$\mathbf{q}_{(\cdot)}$	Quantisierungsfehler des Symbols $(\cdot)$, $\in \mathbb{R}^{N_s}$
$\underline{P}_{\kappa_c}$	Logarithmierte Wahrscheinlichkeit für Codebit C_{κ_c}
$\sigma^2_{(\cdot)}$	Varianz des Quantisierungsrauschens
σ_0^2	Varianz des Rauschens
s	Zustand des Encoders
s'	Folgezustand des Encoders
s_k	Modulationssymbol mit Index k
$s(t)$	Zeitkontinuierliches Sendesignal
$\mathbf{s}$	Sendesymbole, $\in \mathbb{C}^{N_s}$
$\overline{\mathbf{s}}$	Geschätzte Sendesymbole, $\in \mathbb{C}^{N_s}$
$\mathcal{S}$	Symbolalphabet
S_m	Modulationssymbol m des Symbolalphabets $\mathcal{S}$
$\text{sinc}(\cdot)$	Sinus Cardinalis, $\frac{\sin(x)}{x}$
r	Coderate
$r(t)$	Zeitkontinuierliches Empfangssignal (Beobachtung)
r_k	Empfangssymbol mit Index k (Beobachtung)
$\mathbf{r}$	Empfangssymbole (Beobachtung), $\in \mathbb{C}^{N_s}$
$\text{Re}\{\cdot\}$	Realteil
$\mathbb{R}$	Menge der reellen Zahlen
τ	Zeitliche Verzögerung der Mehrwegepfade
T_a	Abtastzeit
T_s	Symboldauer
υ	Vorfaktor Soft Interference Cancellation (s. Gl. (4.11))
φ	Mittlere Leistung von $\overline{\mathbf{s}}$ (s. Gl. (4.11))
$\mathbf{\Psi}$	MMSE-Entzerrermatrix (s. Gl. (4.10)), $\in \mathbb{C}^{N_s \times N_s}$
$\text{Var}\{\cdot\}$	Varianz
w_k	Rauschen mit Index k für äquivalentes AWGN-Modell (s. Abs. 4.2.3)
z_k	Entzerrtes Symbol mit Index k
$\mathbf{z}$	Entzerrte Symbole, $\in \mathbb{C}^{N_s}$

Literaturverzeichnis

[ACV14] *Altera - Cyclone V Device Overview.* `http://www.altera.com/literature/lit-cyclone-v.jsp`. Version: Juli 2014

[ADC14] *Texas Instruments: 12-/14-Bit, 160/250MSPS, Ultralow-Power ADC, Data Sheet.* `http://www.ti.com/product/ads4149`. Version: Juli 2014

[AIP14] *Altera - FFT MegaCore Function.* `http://www.altera.com/products/ip/dsp/transforms/m-ham-fft.html`. Version: Juli 2014

[Amd67] Amdahl, G.M.: Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In: *Joint Computer Conference AFIPS*, 1967, S. 483–485

[ATM03] Abe, T. ; Tomisato, S. ; Matsumoto, T.: A MIMO Turbo Equalizer for Frequency-Selective Channels With Unknown Interference. In: *IEEE Transactions on Vehicular Technology* 52 (2003), Mai, Nr. 3, S. 476–482

[BCJR74] Bahl, L. ; Cocke, J. ; Jelinek, F. ; Raviv, J.: Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate. In: *IEEE Transactions on Information Theory* 20 (1974), März, Nr. 2, S. 284–287

[BDMP98] Benedetto, S. ; Divsalar, D. ; Montorsi, G. ; Pollara, F.: Serial Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding. In: *IEEE Transactions on Information Theory* 44 (1998), Nr. 3, S. 909–926

[BGG03] Boutillon, E. ; Gross, W.J. ; Gulak, P.G.: VLSI Architectures for the MAP Algorithm. In: *IEEE Transactions on Communications* 51 (2003), Februar, Nr. 2, S. 175–185

[BGT93] Berrou, C. ; Glavieux, A. ; Thitimajshima, P.: Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes (1). In: *IEEE International Conference on Communications, Technical Program, Conference Record* Bd. 2, 1993, S. 1064–1070

[BH02] Binder, K. ; Heermann, D.W.: *Monte Carlo Simulation in Statistical Physics*. 4. Auflage. Springer, 2002

[BLL04] Bidan, R. L. ; Loat, C. ; Leroux, D.: Real-time MMSE Turbo-Equalization on the TMS320C5509 Fixed-Point DSP. In: *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2004, S. V325–328

[Bos13] Bossert, M.: *Kanalcodierung*. 2. Auflage. Oldenbourg, 2013

[Bri99] Brink, S. T.: Convergence of Iterative Decoding. In: *Electronics Letters* 35 (1999), Nr. 10, S. 806–808

[BSM01] Bronstein, I.N. ; Semedjajew, K.A. ; Musiol, G.: *Taschenbuch der Mathematik*. 5. Auflage. Harri Deutsch, 2001

[CT65] Cooley, J. ; Tukey, J.: An Algorithm for the Machine Calculation of Complex Fourier Series. In: *Mathematics of Computation* 19 (1965), Nr. 90, S. 297–301

[CTB98] Caire, G. ; Taricco, G. ; Biglieri, E.: Bit-Interleaved Coded Modulation. In: *IEEE Transactions on Information Theory* 44 (1998), Nr. 3, S. 927–946

[DJB+95] Douillard, C. ; Jezequel, M. ; Berrou, C. ; Didier, P. ; Picart, A.: Iterative Correction of Intersymbol Interference: Turbo-Equalization. In: *European Transactions on Telecommunications* 6 (1995), September, Nr. 5, S. 507–512

[EPG94] Erfanian, J. ; Pasupathy, S. ; Gulak, P.G.: Reduced Complexity Symbol Detectors with Parallel Structure for ISI Channels. In: *IEEE Transactions on Communications* 42 (1994), Nr. 234, S. 1661–1671

[Eri08] Erickson, J.: *Hacking: The Art of Exploitation*. 2. Auflage. No Starch Press, 2008

[Ett14] *Ettus Research*. `http://www.ettus.com/`. Version: Juli 2014

[FABSE02] Falconer, D. ; Ariyavisitakul, S.L. ; Benyamin-Seeyar, A. ; Eidson, B.: Frequency Domain Equalization for Single-Carrier Broadband Wireless Systems. In: *IEEE Communications Magazine* 40 (2002), Nr. 4, S. 58–66

[Fal11] Falconer, D.: History of Equalization 1860-1980. In: *IEEE Communications Magazine* 49 (2011), Nr. 10, S. 42–50

[FBC07] Fertonani, D. ; Barbieri, A. ; Colavolpe, G.: Reduced-Complexity BCJR Algorithm for Turbo Equalization. In: *IEEE Transactions on Communications* 55 (2007), November, Nr. 11, S. 2224–2224

[FFT14] *Fastest Fourier Transform in the West (FFTW)*. `http://www.fftw.org/`. Version: Juli 2014

[FJ05] Frigo, M. ; Johnson, S.G.: The Design and Implementation of FFTW3. In: *Proceedings of the IEEE* 93 (2005), Februar, Nr. 2, S. 216–231

[Flo63] Flores, I.: *The Logic of Computer Arithmetic*. Prentice Hall, 1963

[Ger10] Gerthsen, C. ; Meschede, D. (Hrsg.): *Gerthsen Physik (Springer-Lehrbuch)*. 24. Auflage. Springer Berlin, 2010

[GLL97] Glavieux, A. ; Laot, C. ; Labat, J.: Turbo-Equalization over a Frequency Selective Channel. In: *International Symposium on Turbo Codes and Related Topics*, 1997, S. 96–102

[Gol05] Goldsmith, A.: *Wireless Communications*. Cambridge University Press, 2005

[GPL14] *GNU General Puplic License*. `http://www.gnu.org/licenses/gpl-3.0.txt`. Version: Juli 2014

[Gro11] Grossmann, M.: Outage Performance Analysis and Code Design for Three-Stage MMSE Turbo Equalization in Frequency-Selective Rayleigh Fading Channels. In: *IEEE Transactions on Vehicular Technology* 60 (2011), Februar, Nr. 2, S. 473–484

[Gro12] Grossmann, M.: *Transmission Strategies for Broadband Wireless Systems with MMSE Turbo Equalization*. Universitätsverlag Ilmenau, 2012

[Gus88] Gustafson, J.L.: Reevaluating Amdahl's Law. In: *Communications of the ACM* 31 (1988), S. 532–533

[Gut10] Gut, A.: *Probability: A Graduate Course (Springer Texts in Statistics)*. Springer, 2010

[Hag88] Hagenauer, J.: Rate-Compatible Punctured Convolutional Codes (RCPC Codes) and their Applications. In: *IEEE Transactions on Communications* 36 (1988), Nr. 4, S. 389–400

[HH89] Hagenauer, J. ; Hoeher, P.: A Viterbi Algorithm with Soft-Decision Outputs and its Applications. In: *IEEE Global Telecommunications Conference and Exhibition* Bd. 3, 1989, S. 1680–1686

[HJB84] Heideman, M. ; Johnson, D.H. ; Burrus, C.S.: Gauss and the History of the Fast Fourier Transform. In: *IEEE ASSP Magazine* 1 (1984), Oktober, Nr. 4, S. 14–21

[HL09] Hoffmann, S. ; Lienhart, R.: *OpenMP: Eine Einführung in die parallele Programmierung mit C/C++*. Springer, 2009

[HLY02] Hanzo, L. L. ; Liew, T. H. ; Yeap, B. L.: *Turbo Coding, Turbo Equalisation and Space-time Coding for Transmission Over Fading Channels*. Wiley-Blackwell, 2002

[Ii714] *Desktop 4th Generation Intel Core Processor Family, Desktop Intel Pentium Processor Family, and Desktop Intel Celeron Processor Family, Datasheet.* `http://www.intel.com/content/www/us/en/processors/core/CoreTechnicalResources.html`. Version: Juli 2014

[Int94] International Telecommunication Union: International Telephone connections and Circuits – General Recommendations on the Transmission Quality for an Entire International Telephone Connection: One-Way Transmission Time. In: *G.114 ITU-T Recommendation* (1994), Mai

[Isa09] Isaacson, C.: *Software Pipelines and SOA: Releasing the Power of Multi-Core Processing.* Addison-Wesley Professional, 2009

[JMW02] Jondral, F.K. ; Machauer, R. ; Wiesler, A.: *Software Radio.* J. Schlembach Fachverlag, 2002

[JW02] Jondral, F.K. ; Wiesler, A.: *Wahrscheinlichkeitsrechnung und stochastische Prozesse: Grundlagen für Ingenieure und Naturwissenschaftler.* 2. Auflage. Vieweg + Teubner, 2002

[Kam04] Kammeyer, K.-D.: *Nachrichtenübertragung.* 3. Auflage. Teubner, 2004

[Kay93] Kay, S.: *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory.* Prentice Hall, 1993

[KK06] Kammeyer, K.-D. ; Kroschel, K.: *Digitale Signalverarbeitung.* 6. Auflage. Teubner, 2006

[KM07] Kansanen, K. ; Matsumoto, T.: An Analytical Method for MMSE MIMO Turbo Equalizer EXIT Chart Computation. In: *IEEE Transactions on Wireless Communications* 6 (2007), Januar, Nr. 1, S. 59–63

[Kro04] Kroschel, K.: *Statistische Informationstechnik: Signal- und Mustererkennung, Paramter- und Signalschätzung.* 4. Auflage. Springer, 2004

[KST04] Koetter, R. ; Singer, A.C. ; Tüchler, M.: Turbo Equalization. In: *IEEE Signal Processing Magazine* 21 (2004), Januar, Nr. 1, S. 67–80

[KWW08] Kashif, F.M. ; Wymeersch, H. ; Win, M.Z.: Monte Carlo Equalization for Nonlinear Dispersive Satellite Channels. In: *IEEE Journal on Selected Areas in Communications* 26 (2008), Februar, Nr. 2, S. 245–255

[LB06] Lopes, R.R. ; Barry, J.R.: The Soft-Feedback Equalizer for Turbo Equalization of Highly Dispersive Channels. In: *IEEE Transactions on Communications* 54 (2006), Mai, Nr. 5, S. 783–788

[LTE14] *3GPP, The Mobile Broadband Standard.* `http://www.3gpp.org/`. Version: Juli 2014

[MB00] Montorsi, G. ; Benedetto, S.: Design of Fixed-Point Iterative Decoders for Concatenated Codes with Interleavers. In: *IEEE Global Telecommunications Conference* Bd. 2, 2000, S. 801–806

[MB07] Meyer-Baese, U.: *Digital Signal Processing with Field Programmable Gate Arrays (Signals and Communication Technology).* 3. Auflage. Springer, 2007

[MBJ06] Muller, O. ; Baghdadi, A. ; Jezequel, M.: Exploring Parallel Processing Levels for Convolutional Turbo Decoding. In: *Information and Communication Technologies* Bd. 2, 2006, S. 2353–2358

[Men97] Mengali, U.: *Synchronization Techniques for Digital Receivers (Applications of Communications Theory).* Springer, 1997

[MGM+11] Munshi, A. ; Gaster, B. ; Mattson, T.G. ; Fung, J. ; Ginsburg, D.: *OpenCL Programming Guide.* Addison-Wesley Professional, 2011

[Mit92] Mitola, III J.: Software Radios - Survey, Critical Evaluation and Future Directions. In: *National Telesystems Conference*, 1992, S. 13/15–13/23

[MPRZ99] Masera, G. ; Piccinini, G. ; Roch, M.R. ; Zamboni, M.: VLSI Architectures for Turbo Codes. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 7 (1999), September, Nr. 3, S. 369–379

[NCU14] *NVIDIA CUDA, Parallel Programming and Computing Platform*. `http://www.nvidia.com/object/cuda_home_new.html`. Version: Juli 2014

[NFM09] Novak, C. ; Fertl, P. ; Matz, G.: Quantization for Soft-Output Demodulators in Bit-Interleaved Coded Modulation Systems. In: *IEEE International Symposium on Information Theory*, 2009, S. 1070–1074

[NGG14] *NVIDIA GeForce GTX 760, Specifications*. `http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-760/specifications`. Version: Juli 2014

[Nut14] *Nutaq*. `http://www.nutaq.com/`. Version: Juli 2014

[OCL14] *OpenCL - The Open Standard for Parallel Programming of Heterogeneous Systems*. `http://www.khronos.org/opencl/`. Version: Juli 2014

[OCo14] *OpenCores*. `http://opencores.org/`. Version: April 2014

[OMP14] *The OpenMP API Specification for Parallel Programming*. `http://openmp.org`. Version: Juli 2014

[PCFB10] Peng, R. ; Chen, R. ; Farhang-Boroujeny, B.: Markov Chain Monte Carlo Detectors for Channels With Intersymbol Interference. In: *IEEE Transactions on Signal Processing* 58 (2010), April, Nr. 4, S. 2206–2217

[Pet94] Petersen, J.: Implementierungsaspekte zur Symbol-by-Symbol MAP Decodierung von Faltungscodes. In: *ITG-Fachbericht 130* (1994), Oktober, S. 41–48

[PM99] Pätzold, M. ; Mildenberger, O.: *Mobilfunkkanäle. Modellierung, Analyse und Simulation*. Vieweg, 1999

[Prt93] IEEE Standard for Information Technology - Portable Operating System Interface (POSIX) - System Application Program Interface (API) Amendment 1: Realtime Extension (C language). In: *IEEE Std 1003.1b-1993* (1993)

[PS07] Proakis, J. ; Salehi, M.: *Digital Communications*. 5. Auflage. McGraw-Hill, 2007

[Pth95] IEEE Standard for Information Technology - Portable Operating System Interface (POSIX) - System Application Program Interface (API) Amendment 2: Threads Extension (C Language). In: *IEEE Std. 1003.1c-1995* (1995)

[Ree02] Reed, J.H.: *Software Radio: A Modern Approach to Radio Engineering*. Prentice Hall, 2002

[RG75] Rabiner, L. R. ; Gold, B.: *Theory and Application of Digital Signal Processing*. Prentice Hall, 1975

[RMO13] Rondeau, T. ; McCarthy, N. ; O'Shea, T.: SIMD Programming in GNU Radio: Maintainable and User-Friendly Algorithm Optimization with VOLK. In: *Wireless Innovation Forum Conference on Wireless Communication Technologies and Software Defined Radio*, 2013

[Roh85] Rohde, U. L.: Digital HF Radio: A Sampling of Techniques. In: *Third International Conference on HF Communication Systems and Techniques*, 1985

[RVH95] Robertson, P. ; Villebrun, E. ; Höher, P.: A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain. In: *IEEE International Conference on Communications*, 1995, S. 1009–1013

[SFS11] Studer, C. ; Fateh, S. ; Seethaler, D.: ASIC Implementation of Soft-Input Soft-Output MIMO Detection Using MMSE Parallel Interference Cancellation. In: *IEEE Journal of Solid-State Circuits* 46 (2011), Juli, Nr. 7, S. 1754–1765

[SIM14] *Simulink - Simulation and Model-based Design.* `http://www.mathworks.de/products/simulink/`. Version: Juli 2014

[SJ13] Schwall, M. ; Jondral, F.K.: High-Speed Turbo Equalization for GPP-Based Software Defined Radios. In: *IEEE Military Communications Conference*, 2013, S. 1592–1596

[SKJ95] Sari, H. ; Karam, G. ; Jeanclaude, I.: Transmission Techniques for Digital Terrestrial TV Broadcasting. In: *IEEE Communications Magazine* 33 (1995), Nr. 2, S. 100–109

[SNJ12] Schwall, M. ; Nagel, S. ; Jondral, F.K.: Code Parallelization for Multi-Core Software Defined Radio Platforms with OpenMP. In: *Journal of Signal Processing Systems* 69 (2012), Nr. 1, S. 67–74

[STJ13] Schwall, M. ; Thill, R. ; Jondral, F.K.: Parallelization Strategies for Multi-Core General Purpose Processors. In: *Virginia Tech Symposium on Wireless Communications*, 2013

[TEE10] Teege, G. ; Eggendorfer, T. ; Eiseler, V.: *Militärische Funkkommunikation.* Books on Demand, 2009-2010

[TKS02] Tüchler, M. ; Koetter, R. ; Singer, A.C.: Turbo Equalization: Principles and New Results. In: *IEEE Transactions on Communications* 50 (2002), Mai, Nr. 5, S. 754–767

[TS11] Tüchler, M. ; Singer, A.: Turbo Equalization: An Overview. In: *IEEE Transactions on Information Theory* 57 (2011), Februar, Nr. 2, S. 920–952

[TSK02] Tüchler, M. ; Singer, A.C. ; Koetter, R.: Minimum Mean Squared Error Equalization Using A Priori Information. In: *IEEE Transactions on Signal Processing* 50 (2002), Nr. 3, S. 673–683

[TV05] Tse, D. ; Viswanath, P.: *Fundamentals of Wireless Communication.* Cambridge University Press, 2005

[Tü04] Tüchler, M.: Design of Serially Concatenated Systems Depending on the Block Length. In: *IEEE Transactions on Communications* 52 (2004), Nr. 2, S. 209–218

[Vit67] Viterbi, A.: Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. In: *IEEE Transactions on Information Theory* 13 (1967), April, Nr. 2, S. 260–269

[VOL14] *Vector-Optimized Library of Kernels (VOLK).* `http://gnuradio.org/redmine/projects/gnuradio/wiki/Volk/`. Version: Juli 2014

[WNW14] *Joint Tactical Networking Center - Our Products.* `jtnc.mil/Pages/Products.aspx`. Version: Juli 2014

[WP99] Wang, X. ; Poor, H.V.: Iterative (Turbo) Soft Interference Cancellation and Decoding for Coded CDMA. In: *IEEE Transactions on Communications* 47 (1999), Juli, Nr. 7, S. 1046 –1061

[WP13] Wyglinski, A.M. ; Pu, D.: *Digital Communication Systems Engineering with Software-Defined Radio.* Artech House, 2013

[WS73] Walzman, T. ; Schwartz, M.: Automatic Equalization Using the Discrete Frequency Domain. In: *IEEE Transactions on Information Theory* 19 (1973), Nr. 1, S. 59–68

[WSC10] Wu, M. ; Sun, Y. ; Cavallaro, J.R.: Implementation of a 3GPP LTE Turbo Decoder Accelerator on GPU. In: *IEEE Workshop on Signal Processing Systems*, 2010, S. 192–197

[XIP14] *Xilinx - LogiCORE IP Fast Fourier Transform v7.1.* `http://www.xilinx.com/products/intellectual-property/FFT.htm`. Version: Juli 2014

[XIS14] *Xilinx - ISE Design Suite.* `http://www.xilinx.com/products/design-tools/ise-design-suite/`. Version: Juli 2014

[XK714] *Xilinx - 7 Series FPGAs Overview*. http://www.xilinx.com/products/silicon-devices/fpga/virtex-7/. Version: Juli 2014

[XS614] *Xilinx - Spartan-6 Family Overview*. http://www.xilinx.com/products/silicon-devices/fpga/spartan-6/. Version: Juli 2014

[XSG14] *Xilinx - System Generator for DSP*. http://www.xilinx.com/tools/sysgen.htm. Version: Juli 2014

[XV614] *Xilinx - Virtex-6 Family Overview*. http://www.xilinx.com/products/silicon-devices/fpga/virtex-6/. Version: Juli 2014

[YC12] Yoge, D.R.N. ; Chandrachoodan, N.: GPU Implementation of a Programmable Turbo Decoder for Software Defined Radio Applications. In: *International Conference on VLSI Design*, 2012, S. 149–154

[YW99] Yufei, W. ; Woerner, B.D.: The Influence of Quantization and Fixed Point Arithmetic Upon the BER Performance of Turbo Codes. In: *IEEE Vehicular Technology Conference* Bd. 2, 1999, S. 1683–1687

[ZP04] Zhang, Y. ; Parhi, K.K.: Parallel Turbo decoding. In: *IEEE International Symposium on Circuits and Systems* Bd. 2, 2004, S. II–509–12 Vol.2

Studentische Arbeiten

Jesko Jochum	*Entwurf einer universellen Sender- und Empfänger-Implementierung für einen FPGA*, Bachelorarbeit, Karlsruher Institut für Technologie, 2010
Stevan Đorđević	*Untersuchung und Bewertung von paralleler Programmierung für Multicore SDR Plattformen mit OpemMP*, Bachelorarbeit, Karlsruher Institut für Technologie, 2011
Armel Wabo Kogaing	*Programmierung und Implementierung eines Viterbi-Decoders auf einem DSP*, Studienarbeit, Karlsruher Institut für Technologie, 2011
Christian Reimer	*Aufbau eines Demonstrators zur Synchronisation einer digitalen Übertragungsstrecke*, Bachelorarbeit, Karlsruher Institut für Technologie, 2011
Jan Krämer	*Implementierung einer Kanalschätzung und Entzerrung für SC-FDE*, Bachelorarbeit, Karlsruher Institut für Technologie, 2011
Fabian Witt	*Synchronisation einer Single Carrier Frequency Domain Equalization Übertragung*, Bachelorarbeit, Karlsruher Institut für Technologie, 2011
Stefan Daniel Lentz	*Optimierte Realisierung eines Turbo-Entzerrers*, Bachelorarbeit, Karlsruher Institut für Technologie, 2012

Lingarao Chandrakani	*MAC Layer Design for a MANET, Masterarbeit*, Karlsruher Institut für Technologie, 2012
Khaled Nesrallah	*Portierung einer Wellenform auf eine heterogene SDR Plattform*, Studienarbeit, Karlsruher Institut für Technologie, 2012
Jonas Pfau	*Optimierte Implementierung eines Turbo-Entzerrers auf einem FPGA*, Masterarbeit, Karlsruher Institut für Technologie, 2013
David Leuck	*Turbo Entzerrung: Untersuchung von Quantisierungseffekten*, Bachelorarbeit, Karlsruher Institut für Technologie, 2013
Ruben Thill	*Turbo Entzerrung: Leistungssteigerung durch Pipelining*, Bachelorarbeit, Karlsruher Institut für Technologie, 2013
Christian Reimer	*Turbo Entzerrung für GMSK-modulierte Signale*, Masterarbeit, Karlsruher Institut für Technologie, 2013
Tobias Schwegler	*Analyse suboptimaler Turbo-Entzerrer bei nicht-perfekter Kanalkenntnis*, Masterarbeit, Karlsruher Institut für Technologie, 2013
Moritz Fischer	*Design and Implementation of a Cache Coherent Co-Processor for Embedded SDR Platforms*, Masterarbeit, Karlsruher Institut für Technologie, 2014
Jan Krämer	*Analyse der Parallelisierbarkeit eines MAP-Decoders für Vielkernprozessoren*, Masterarbeit, Karlsruher Institut für Technologie, 2014
Stefan Daniel Lentz	*Turbo-Entzerrung für FBMC-Verfahren*, Masterarbeit, Karlsruher Institut für Technologie, 2014
Johannes Kattinger	*Aufbau eines Funkpeilsystems mit GNU Radio*, Bachelorarbeit, Karlsruher Institut für Technologie, 2014

Stipendium

Teile dieser Arbeit entstanden während eines Forschungsaufenthalts an der University of Arizona in Tucson, USA. Dieser Aufenthalt wurde vom Karlsruhe House of Young Scientists (KHYS) und dem Communications Engineering Lab (CEL) gefördert.

Ich bedanke mich herzlich für diese Förderung und das mir entgegengebrachte Vertrauen.

Index