Effiziente Entwurfsverfahren zur hardwarebasierten Signalverarbeitung elementarer Funktionen für die drahtlose Kommunikation

Dem Fachbereich Physik, Elektrotechnik und Informationstechnik
der Universität Bremen

zur Erlangung des akademischen Grades eines
DOKTOR-INGENIEUR (Dr.-Ing.)
genehmigte Dissertation

von
Dipl.-Ing. Jochen Rust
aus Lehrte

Referent:	Professor Dr.-Ing. St. Paul
Korreferentin :	Professorin Dr.-Ing. D. Göhringer
Eingereicht am:	28.01.2014
Tag des Promotionskolloquiums:	10.03.2014

Kurzfassung

Drahtlose Kommunikationssysteme stellen eine Schlüsseltechnologie aktueller und zukünftiger Ansätze zur Datenübertragung dar, da sie in einer Vielzahl unterschiedlicher Bereiche des täglichen Lebens zum Einsatz kommen. Die Entwicklung der letzten Jahre lässt dabei erkennen, dass der Umfang der zu bewältigenden Aufgaben und Anwendungen deutlich ansteigen wird. So ist z.B. im Mobilfunk eine stetig wachsende Anzahl an Netzteilnehmern zu erkennen, die immer größere Mengen an zu übertragenden Daten beansprucht. Für mobile Endgeräte wird zudem eine möglichst lange Akkulaufzeit gefordert. Um diesen Ansprüchen gerecht zu werden, hat sich die effiziente digitale Signalverarbeitung als vielversprechender Ansatz erwiesen. Hierbei werden anstehende Signalverarbeitungsaufgaben, z.B. die Berechnung von Algorithmen, möglichst optimal und durch direkte Abbildung an die zugrunde liegende Hardware angepasst. Für zukünftige drahtlose Kommunikationssysteme ist jedoch aufgrund der steigenden Algorithmenkomplexität davon auszugehen, dass dieses Vorgehen nicht mehr ausreicht, die hohen Anforderungen zu erfüllen.

Der Schwerpunkt dieser Arbeit liegt auf dem Entwurf von anwendungsspezifischen Schaltungen, die für eine näherungsweise Berechnung elementarer Funktionen optimiert sind. Hierzu wird ein spezielles Approximationsverfahren vorgestellt, das mittels Unterteilung eines gegebenen Funktionsverlaufs sowie Verwendung hardwareoptimierter Geradengleichungen den Signalverarbeitungsaufwand deutlich verringern kann. Durch ein automatisiertes Vorgehen zur Umsetzung der Approximation auf eine äquivalente Hardwarebeschreibung kann das Verfahren zudem beim Entwurf digitaler Schaltungen verwendet werden. Anhand ausgewählter Signalverarbeitungsaufgaben aus den Bereichen drahtlose Sensornetze und Mobilfunk werden verschiedene Funktionsapproximationen erstellt und in die zugehörigen Algorithmen integriert. Neben einer simulativen, modellbasierten Auswertung der durch die Approximation auftretenden Rechenfehler wird für jede betrachtete Anwendung eine angepasste Schaltungsumsetzung vorgestellt. Abschließend werden die erhaltenen Ergebnisse mit aktuellen Referenzen verglichen.

Abstract

Wireless communication is a key aspect of current and future data communication, thus, it is used in several different application areas of daily life. Due to the development in recent years, it is most likely that the overall processing effort in this scope will increase significantly. For example, within the scope of mobile telephony, there is a steady growth of subscribers, which will lead to a higher amount of payload and transceiver activity. For wireless devices, high battery lifetime is also demanded. In order to fulfill the above requirements, efficient digital signal processing has proven itself to be a promising approach. To this end, the signal processing task, e.g. the calculation of algorithms, is directly mapped onto the underlying hardware. However, for future wireless communication systems, this approach will lead to insufficient results, due to the expected increase of algorithmic complexity.

This thesis focusses on approximate signal processing of elementary functions in the context of digital circuit design for wireless communication. An adapted design methodology is presented that uses efficient function segmentation as well as hardware-optimized linear equations, in order to reduce the overall signal processing effort. As the translation of the function approxmiation to an equivalent hardware description is performed automatically, this can be embedded into the digital design flow. In this work, several elementary functions from mobile communication and wireless sensor network applications are approximated by this methodology. Model-based simulation and validation is performed, as well as corresponding hardware implementations are presented for each application. Finally, all results are evaluated by comparison to appropriate references.

Vorwort

Die vorliegende Arbeit befasst sich mit den Forschungsaktivitäten im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Theoretische Elektrotechnik und Mikroelektronik (ITEM) der Universität Bremen im Arbeitsbereich Kommunikationselektronik. Ich möchte mich an dieser Stelle ganz herzlich bei Herrn Professor Steffen Paul für die Ermöglichung und qualifizierte Betreuung dieser Arbeit sowie die zahlreichen wertvollen inhaltlichen und methodischen Anregungen bedanken. Frau Professorin Diana Göhringer von der Ruhr-Universität Bochum gilt mein besonderer Dank für die Übernahme des Koreferats. Herrn Professor Garcia-Ortiz und Herrn Professor Armin Dekorsy danke ich für die Übernahme der Prüfertätigkeit.

Danken möchte ich allen Mitarbeiterinnen und Mitarbeitern am ITEM, die auf die eine oder andere Weise zum Gelingen der Arbeit beigetragen haben. Besonders hervorheben möchte ich hier meine Kollegen der ersten Stunde, Herrn Till Wiegand und Herrn Ole Bischoff, die ihr Fachwissen über ihre jeweiligen Forschungsbereiche immer bereitwillig mit mir geteilt haben. Ich danke meinen Bürokollegen Herrn Sascha Kneip und Herrn Janpeter Höffmann für viele fachliche Gespräche und das gute Miteinander. Weiterer Dank gilt Frau Xinwei Wang, Herrn Jonas Pistor, Herrn Frank Ludwig und Herrn Nils Heidmann für die gute Zusammenarbeit bei Veröffentlichungen sowie Frau Valerie Gerdes für ihre große Hilfsbereitschaft bei administrativen Aufgaben.

Zudem möchte ich mich bei den Studentinnen und Studenten bedanken, die mich mit ihren Arbeiten unterstützt haben, stellvertretend seien an dieser Stelle Herr Max Gröning und Herr Christof Osewold genannt.

Ein ganz besonderer Dank gilt auch meinen Freunden und meiner Familie, die mich all die Jahre großartig unterstützt haben. Allen voran danke ich meiner Partnerin Christine, deren Liebe, Zuspruch und Verständnis mir viel Kraft zur Fertigstellung dieser Arbeit gegeben haben.

Inhaltsverzeichnis

1 Einleitung

Effiziente Verarbeitung von Signalen und Algorithmen stellt eine der bedeutendsten Herausforderungen aktueller und zukünftiger drahtloser Kommunikationssysteme dar: So wird im Bereich Mobilfunk bis 2017 eine Erhöhung des weltweiten Datenaufkommens um das Siebenfache auf ungefähr $11,2$ Exabyte im Monat erwartet [19]. Ein Grund hierfür ist zum Einen die stetig wachsende Zahl an Mobilfunkteilnehmern. Zum Anderen lässt sich dieser Effekt durch steigenden Datenverkehr jedes einzelnen Teilnehmers erklären. Allein für Smartphones wird von einem Wachstum zwischen 2012 und 2017 von ca. 778% ausgegangen (siehe Tab. 1.1), was sich durch die zukünftig massiv verstärkte Nutzung von Diensten mit hoher Netzauslastung, wie *Streaming-* und *Cloud-*Anwendungen, erklären lässt [73]. Für drahtlose Sensornetze zeichnet sich eine ähnliche Entwicklung ab: Trends wie das Internet der Dinge (engl.: *Internet-of-things*, IoT) bedeuten in diesem Zusammenhang, dass Sensorknoten neben herkömmlichen Aufgaben auch die Verarbeitung des TCP/IP-Protokolls bewältigen müssen. Zudem sollen Sensorknoten zunehmend autonom agieren, z.B. durch dynamische Topologiekonfiguration [78]. Sowohl im Mobilfunk als auch für drahtlose Sensornetze geht die zufriedenstellende Bewältigung dieser angesprochenen Anforderungen mit einem signifikanten Anstieg an Signalverarbeitungsaufwand und der hardwarebasierten Abarbeitung von komplexen Algorithmen einher.

Eine weitere wichtige Aufgabe zukünftiger drahtloser Kommunikationssysteme ist die Reduktion des Energieverbrauchs [29], da sich andernfalls die Akkulaufzeit mobiler Netzteilnehmer signifikant verringert. Für den Betrieb von Basisstationen stellt dieses Szenario auch unter ökonomischen Gesichtspunkten eine Herausforderung dar: So ist nach [70] im Jahr 2020 ein wirtschaftlicher Betrieb nur bei deutlicher Minimierung des Energieverbrauchs, ca. um das Zehnfache, möglich.

Zur Bewältigung dieser Anforderungen hat sich der Einsatz anwendungsspezifischer Schaltungen zur effizienten digitalen Signalverarbeitung in den letzten Jahren bewährt. Kern dieses Ansatzes ist der Entwurf von Schaltungen, die an eine spezielle Anwendung angepasst sind, wobei durch verschiedene Kriterien, z.B. die Wahl der Hardwarearchitektur oder Implementierungstechniken, die Performance festgelegt wird. Ein vielbeachteter aktueller Ansatz im Bereich der drahtlosen Kommunikation ist zudem die feh-

lerbehaftete Signalverarbeitung, bei der z.B. algorithmische Terme nur näherungsweise berechnet werden, was bspw. aufgrund rauschbehafteter Übertragungskanäle zulässig ist [39]. Die daraus entstehenden Vereinfachungen in der Signalverarbeitung führen i.d.R. zu deutlichen Steigerungen der Effizienz und wirken somit den vorangehend beschriebenen Einbußen hinsichtlich Datendurchsatz oder Energieverbrauch entgegen.

Generell ist die Umsetzung fehlerbehafteter digitaler Signalverarbeitung durch eine Reihe unterschiedlicher Ansätze möglich. Bei anwendungsspezifischen, algorithmusbasierten Problemstellungen lassen sich i.d.R. mathematische Terme, Gleichungen oder nicht triviale, elementare Funktionen, z.B. Polynome oder trigonometrische Funktionen, erkennen, die aufgrund ihres häufigen Auftretens oder ihres übermäßigen Rechenaufwandes die Performance deutlich begrenzen. Ein vielbeachteter Ansatz zur effizienten Berechnung solcher Funktionen ist die abschnittsweise Funktionsapproximation, weswegen sie in einer Vielzahl unterschiedlicher Anwendungsgebiete eingesetzt wird. Neben der möglichst performanten Signalverarbeitung stellt für diesen Ansatz auch die Erzeugung einer geeigneten Approximation eine Herausforderung dar. Im Kontext des anwendungsspezifischen Schaltungsentwurfs für die drahtlose Kommunikation wurden Funktionsapproximationen bislang fast ausschließlich manuell erstellt, was einen sehr zeitaufwändigen Lösungsansatz, insbesondere für sehr hohe Rechengenauigkeiten, darstellt. Eine automatische Erzeugung zur Näherung elementarer Funktionen wurde hierfür bislang noch nicht erschöpfend untersucht.

1.1 Eigener Beitrag

Kern der Arbeit ist der Entwurf performanter digitaler Schaltungen im Bereich der drahtlosen Kommunikation. Das grundlegende Vorgehen basiert dabei auf der Implementierung approximierter, elementarer Funktionen durch Verwendung eines dafür entwickelten Synthesewerkzeugs zur automatischen linearen Funktionsapproximation (ALFA).

Zur effizienten Implementierung anwendungsspezifischer Probleme muss zunächst ein passender Algorithmus ausgewählt werden. Es lassen sich i.d.R. mehrere unterschiedliche Ansätze zur Lösung einer Signalverarbeitungsaufgabe erkennen, die sich hinsichtlich ihrer Implementierung unterscheiden, z.B. Householder-, Gram-Schmidt- oder Givensbasierte QR-Zerlegung. Je nach Einsatzgebiet und Entwurfskriterien, z.B. Zielplattform der Hardware (ASIC oder FPGA), kann die Eignung besagter Ansätze variieren. Daher ist die Auswahl eines geeigneten Algorithmus ein wichtiger Bestandteil der Arbeit.

Für den Einsatzbereich anwendungsspezifischer Schaltungen lassen sich zudem verschiedene Entwurfsziele, also z.B. hoher Datendurchsatz, geringe Chipfläche oder geringer Energieverbrauch, erkennen, was sich auf die Auswahl der Hardwarearchitektur maßgeblich auswirkt. Ausgewählte Algorithmen können durch geeignete Partitionie-

Tabelle 1.1: Erwartete Entwicklung des durchschnittlichen mobiler Datenverkehrs (in MByte) im Monat bis 2017 für unterschiedliche Gerätearten, verändert nach [19].

Geräteart	2012	2017	Anstieg
M2M-Gerät	64	330	515%
Smartphone	342	2660	778%
4G Smartphone	1302	5114	515%
Tablet-PC	820	5387	657%
Laptop	2503	5731	229%
Sonstige Geräte	6,8	31	455%

rung an entsprechende Vorgaben angepasst werden: Zum Einen müssen sie in Terme, die sich gut in Hardware abbilden lassen, unterteilt werden. Zum Anderen soll die Hardware für jeden Verarbeitungsschritt möglichst gut ausgelastet sein. Beide Forderungen müssen von der verwendeten Hardware möglichst optimal umgesetzt werden.

Im Rahmen dieser Arbeit werden ausgewählte Algorithmen des Mobilfunks bzw. drahtloser Sensornetze betrachtet. Nicht triviale, elementare Funktionen werden durch den eingangs angesprochenen ALFA-Ansatz zeiteffizient in eine passende HDL-Beschreibung übersetzt. Das zugehörige Verfahren basiert auf der Nachbildung eines gegebenen Funktionsverlaufs durch multiplizierenfreie Geradengleichungen, die äußerst geringe algorithmische Komplexität aufweisen. Anhand vorab spezifizierter Parameter, z.B. des mittleren absoluten Fehlers der Approximation, kann die Performance der resultierenden Hardwarebeschreibung variiert werden.

Um Aussagen über die Qualität der entworfenen Schaltungen und die Funktionsapproximationen zu machen, werden die ausgewählten Anwendungen mit aktuellen Referenzen verglichen. Neben einer hardwarebasierten Auswertung der Performance nach logischer und physikalischer Synthese, werden auch algorithmische Aspekte, wie Rechengenauigkeit oder algorithmische Komplexität, betrachtet. Je nach Anwendung stehen unterschiedliche Entwurfskriterien im Vordergrund, wodurch entsprechend Aussagen über die Eignung der ALFA-Synthese getroffen werden können.

1.2 Gliederung

In Kap. 2 werden die wesentlichen Eigenschaften von ASICs vorgestellt. Dies umfasst zunächst eine einleitende Klassifikation, also die Abgrenzung gegenüber anderen Schaltungsarten. Hierauf folgen aktuelle Ansätze zur Implementierung, bezugnehmend auf gängige Hardwarearchitekturen sowie Techniken zur Priorisierung einzelner Entwurfs-

kriterien. Der letzte Abschnitt dieses Kapitels behandelt den Ablauf des Hardwareentwurfs für ASICs.

Kap. 3 dient der Vorstellung der Grundlagen digitaler Signalverarbeitung elementarer Funktionen. Zunächst wird ein Überblick über unterschiedliche Arten von Zahlensystemen und -darstellungen in der Digitaltechnik gegeben. Darauf folgt die Vorstellung der Hardwareumsetzung arithmetischer Basisoperatoren. Für die Berechnung nicht trivialer, elementarer Funktionen werden danach iterative, polynomiale und tabellengestützte Ansätze erläutert, bevor abschließend das grundlegende Prinzip der ALFA-Synthese anhand der zugrunde liegenden Heuristik der Parameterbestimmung sowie der Hardwareübersetzung ausführlich beschrieben wird.

Kap. 4 behandelt das Anwendungsfeld der Prädiktion von Messdaten im Containerumfeld zur energieeffizienten Überwachung der Umweltparameter durch batteriebetriebene Sensorknoten. Es werden zunächst gängige Architektur- und Topologieansätze für drahtlose Sensornetze eingeführt. Hierauf folgt ein Überblick über Prädiktoren, die die Verwendung dynamischer Messintervalle ermöglichen und somit den Energieverbrauch senken können. Nach einer allgemeinen Evaluation werden darauf folgend verschiedene software- bzw. hardwarebasierte Ansätze zur Performancesteigerung programmiert bzw. implementiert und abschließend miteinander verglichen und ausgewertet.

In Kap. 5 werden Algorithmen aktueller Übertragungsverfahren des Mobilfunks untersucht. Der Schwerpunkt liegt dabei auf Mehrantennensystemen und OFDM-Multiplexverfahren. Nach einer Vorstellung der Grundlagen wird die Implementierung eines digitalen Oszillators für die Frequenzoffsetkompensation beschrieben, bei der einzelne Terme durch ALFA-basierte Approximation optimiert sind. Hierauf folgt ein Ansatz zur Berechnung der QR-Zerlegung, die in Mehrantennensystemen in verschiedenen Bereichen Anwendung finden kann. Auch die Implementierung dieses Algorithmus wird durch ALFA optimiert. Die anwendungsspezifische Evaluation beider Ansätze stellt den Abschluss des Kapitels dar.

Im letzten Kapitel werden in Form eines zusammenfassenden Fazits sowohl die anwendungsspezifischen Ergebnisse als auch die Eignung der ALFA-Synthese an sich resümiert und bewertet. Zudem erfolgt ein Ausblick, der mögliche Erweiterungen und Verbesserungen zum vorgestellten ALFA-Ansatz und den einzelnen anwendungsspezifischen Implementierungen behandelt.

2 Anwendungsspezifische integrierte Schaltungen

Anwendungsspezifische integrierte Schaltungen (engl.: *application-specific integrated circuit*, ASIC), auch anwendungsspezifische Schaltungen genannt, werden heutzutage in einer Vielzahl unterschiedlicher Bereiche eingesetzt [13] [55]. Ihr grundlegender physikalischer Aufbau besteht aus einem monolithischen Halbleiterkristall, auf dem metallische Ebenen sowie isolierende Schichten aufgetragen sind [21] [127]. Hierdurch können aktive und passive elektrische Bauelemente mit hoher Integrationsdichte erstellt werden, auf deren Grundlage sich Schaltungen mit geringem Energieverbrauch, geringer Komplexität und hoher Taktfrequenz realisieren lassen. Generell werden solche Schaltungen als Mikrochips, monolithische Schaltungen oder integrierte Schaltungen (engl.: *integrated circuit*, IC) bezeichnet [75].

ASICs stellen eine Untergruppe von ICs dar, die Anforderungen an eine konkrete Funktionalität oder Anwendung erfüllen. Aufgrund dieses beschränkten Einsatzgebietes lassen sich im Vergleich zu herkömmlichen Mehrzweck-Schaltungen (engl:*general-purpose integrated circuit*, GPIC), wie Mikrocontrollern, Standard-ICs oder Operationsverstärkern, bessere Performance, also Effizienz im Bezug auf Energieverbrauch, Komplexität, Latenz und/oder Taktfrequenz, erzielen. Um die komplexe Algorithmik, mit der anwendungsspezifische Signalverarbeitung vielfach einhergeht, zu bewältigen, wird i.d.R. auf eine digitale Schaltungsumsetzung zurückgegriffen.

In diesem Kapitel werden ASICs zunächst allgemein klassifiziert. Danach folgt die Vorstellung gängiger Hardwareumsetzungen, bevor ein Überblick über den ASIC-Hardwareentwurf den Abschluss dieses Kapitels bildet.

2.1 Klassifizierung von ASICs

Generell ist die Klassifizierung von ICs unter vielen verschiedenen Gesichtspunkten möglich. Um die Besonderheiten von ASICs hervorzuheben, haben sich entwicklungs- und komplexitätsorientierte Klassifikationen als sinnvoll erwiesen, weswegen diese Aspekte im Folgenden näher erläutert werden.

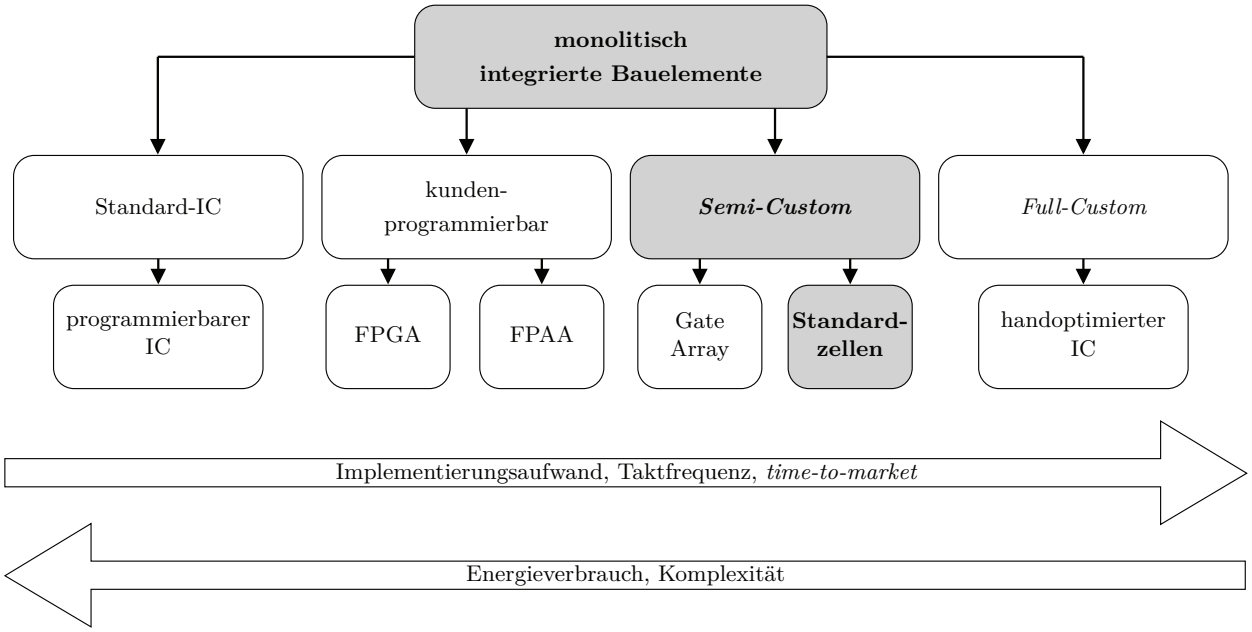


Abbildung 2.1: Entwicklungsorientierte IC-Klassifizierung, verändert nach [18] und [126]. Die grau unterlegten Boxen markieren die Einordnung der in dieser Arbeit schwerpunktmäßig behandelten Verfahren zum ASIC-Entwurf (siehe Kap. 2.4). Anhand der horizontalen Pfeile lässt sich die Zu- und Abnahme einzelner Entwurfskriterien (siehe Kap. 2.4.1) erkennen.

Entwicklungsorientierte Klassifizierung Bei entwicklungsorientierter Klassifizierung findet eine Bewertung von Schaltungen nach Herstellungsaufwand, also der Zeit bis zur Marktreife des gesamten Systems (engl.: *time-to-market*), statt, die direkten Einfluss auf die Entwurfskriterien (siehe Kap. 2.4.1) hat. Nach [18] werden ausgehend vom grundlegenden integrierten Schaltungsaufbau ICs als

- Standard-ICs,
- kundenprogrammierbare Schaltungen,
- *Semi-Custom* Schaltungen und
- *Full-Custom* Schaltungen

klassifiziert (siehe Abb. 2.1).

Der ASIC-Entwurf ist i.d.R. dem standardzellenbasierten Hardwareentwurf, der zu den *Semi-Custom* Schaltungen gehört, zuzuordnen [13]. Das grundlegende Prinzip beruht bei diesem Verfahren auf der Verwendung vordefinierter elementarer Bauelemente, sogenannter Standardzellen. Diese umfassen u.a. neben Gattern und Flipflops auch

Tabelle 2.1: Klassifizierung durch Anzahl der verwendeten Gatteräquivalente nach [55]. Die fett gedruckten Einträge kennzeichnen den im Rahmen dieser Arbeit relevanten Bereich.

Komplexität	Abkürzung	Gatteräquivalente + 1 Bit Speicher
Small-Scale integration	SSI	1 - 10
Medium-Scale integration	MSI	10 - 100
Large-Scale integration	**LSI**	**100 - 10.000**
Very-Large-Scale integration	**VLSI**	**10.000 - 1.000.000**
Ultra-Large-Scale integration	ULSI	$\geq$ 1.000.000

komplexere Strukturen wie Multiplexer oder Volladdierer. Durch Modellierung und Abstraktion des Verhaltens dieser Standardzellen auf unterschiedlichen Hardwareentwurfsebenen ist zum Einen ein zeiteffizienter Schaltungsentwurf möglich. Zum Anderen sind hierdurch, aufgrund der zumeist hohen Anzahl an unterschiedlichen Standardzellen, vielseitige Verhaltensbeschreibungen möglich.

Komplexitätsorientierte Klassifikation Komplexitätsorientierte Klassifikation dient der Bewertung von ICs hinsichtlich der benötigten Chipfläche [116]. Eine einfache Methode zur Größenabschätzung ist durch Betrachtung des Integrationsgrades einer Schaltung, also der Komplexität, möglich, wofür bei digitalen Schaltungen i.d.R. die Anzahl verwendeter Logikgatter zur Bewertung herangezogen wird. Als Referenz dient dabei i.d.R. ein einfaches NAND-Gatter [55]. Die ermittelte Anzahl an Gattern wird in Gatteräquivalenten (engl.: *gate equivalent*, GE) angegeben. Diese Vorgehensweise stellt somit eine einfache Möglichkeit zur Abschätzung und Bewertung der Größe eines digitalen ICs dar, ohne dass prozessbedingte Faktoren wie Strukturgröße oder physikalische Anordnung der Bauelemente berücksichtigt werden müssen. Eine gängige Gruppierung hinsichtlich des Integrationsgrades nach [55] ist in Tab. 2.1 gegeben.

2.2 Hardwarearchitekturen

Ein wichtiges Kriterium des ASIC-Entwurfs ist die Auswahl der bestgeeigneten Hardwarearchitektur für die umzusetzende Anwendung oder den umzusetzenden Algorithmus. Dabei existiert theoretisch eine unbegrenzte Anzahl an Hardwarearchitekturen. In der Praxis hat sich aber ein fester Satz an gängigen Ansätzen bewährt. Nachfolgend werden die im Rahmen dieser Arbeit relevanten Architekturen (festverdrahtete Schaltungen, Prozessoren) sowie weitere aktuelle Ansätze vorgestellt.

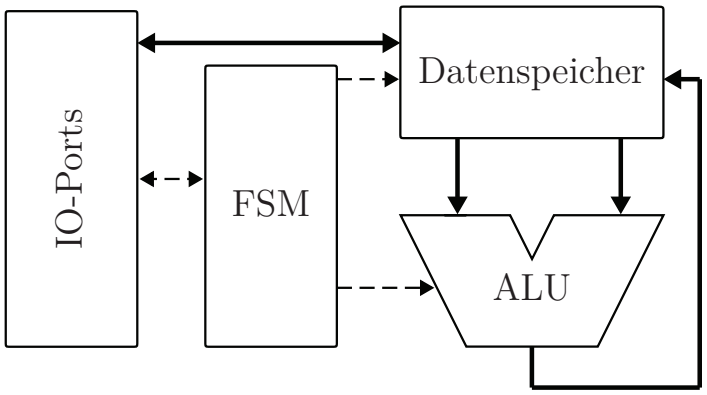


Abbildung 2.2: Prinzipieller Aufbau einer festverdrahteten Schaltung mit FSM-gesteuertem Datenpfad und den IO-Ports als externe Schnittstelle. Die durchgezogenen bzw. gestrichelten Linien markieren Verbindungen für den Austausch von Nutz- bzw. Steuerdaten.

Festverdrahtete Schaltungen Festverdrahtete Schaltungen stellen die direkte Umsetzung von Anwendungen oder Algorithmen in Hardware dar [55], wobei einzelne Funktionsteile durch Abbildung auf entsprechende Hardwareelemente realisiert werden. Sie erreichen i.d.R. bestmögliche Ergebnisse hinsichtlich Komplexität, Energieverbrauch und Taktfrequenz, weswegen sie in vielen Bereichen, z.B. im Mobilfunk [99], als Hardwarebeschleuniger zur Umsetzung anwendungsspezifischer Aufgaben verwendet werden. Ein Nachteil dieser Hardwarearchitektur ist, dass nachträgliche Verhaltensänderungen nicht möglich sind. So muss hierbei die Schaltung zumeist vollständig neu konzipiert, entworfen und gefertigt werden, was mit hohen finanziellen sowie zeitlichen Kosten einhergeht. Festverdrahtete Schaltungen lassen sich somit sinnvoll für Algorithmen verwenden, die regulär sind oder die keine übermäßigen Veränderungen des Verhaltens für gängige Eingangsdaten aufweisen.

Ein übliches Verfahren bei der Umsetzung festverdrahteter Schaltungen ist der Einsatz steuerbarer Datenpfade, bei denen ein Steuerwerk zur Anordnung der vorhandenen Operatoren des Datenpfades sowie der Quell- und Zieloperanden verwendet wird [75]. Einem Steuerwerk liegen zumeist hardwareäquivalente Implementierungen von Zustands- oder Transitionsdiagrammen, die i.d.R. als endliche Zustandsautomaten (engl.: *finite state machine*, FSM) realisiert sind, zugrunde. Dieses Vorgehen erlaubt somit mittels Anwendung des in Kap. 2.3 vertieften Zeitmultiplex-Verfahrens die Verwendung einzelner Hardwaremodule im Datenpfad für unterschiedliche Teile eines Algorithmus. Die Abarbeitungssequenz ist dabei fest vorgegeben und kann nicht umprogrammiert werden. Eine Übersicht über die allgemeine Architektur festverdrahteter Schaltungen ist in Abb. 2.2 gegeben.

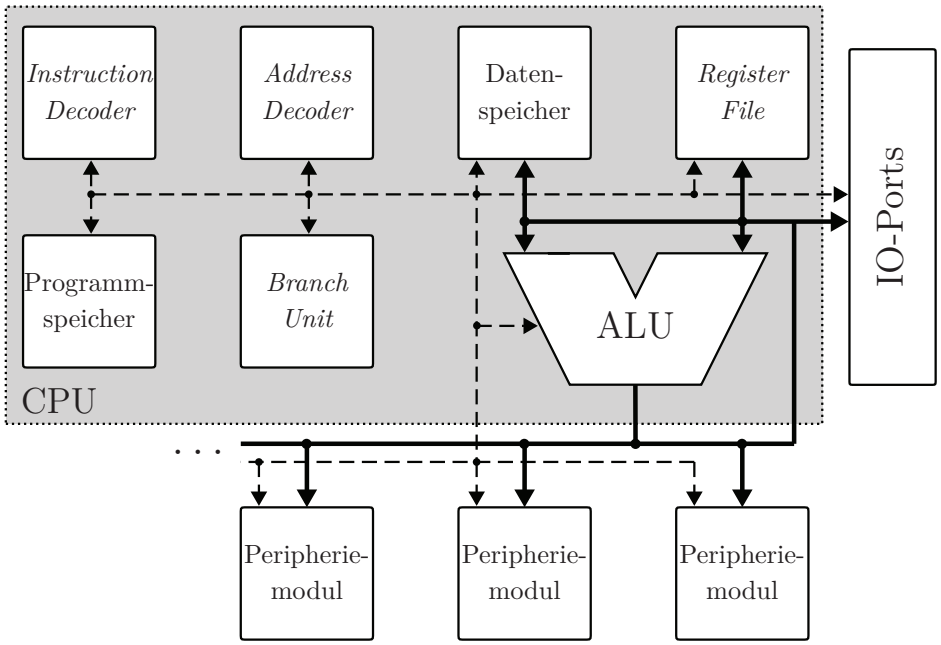


Abbildung 2.3: Prinzipieller Aufbau eines Prozessors, der sich grundsätzlich in Peripherie- und CPU-Module unterteilen lässt. Die IO-Ports bilden die externe Schnittstelle. Die durchgezogenen bzw. gestrichelten Linien markieren Verbindungen für den Austausch von Nutz- bzw. Steuerdaten. Eine detaillierte allgemeine Erläuterung sämtlicher Module findet sich in [83].

Prozessoren Eine weitere wichtige Hardwarearchitektur stellen Prozessoren dar, die sich aus Prozessorkern (engl.: *central processing unit*, CPU) und Peripheriemodulen zusammensetzen [83]. Die CPU besteht dabei zum Einen aus einer komplexen Ansteuereinheit zur Konfiguration des Datenpfades. Die Abarbeitung findet zudem softwarebasiert statt, d.h. der Algorithmus wird durch prozessorspezifische Instruktionen als Programm abgearbeitet. Der Befehlssatz einer solchen Architektur ist für ASICs an eine spezielle Anwendung oder einen speziellen Algorithmus angepasst. Eine solche Schaltung wird auch Prozessor mit anwendungsspezifischem Befehlssatz (engl.: *application-specific instruction-set processor*, ASIP) genannt [109]. Auch für diesen Schaltungstyp lassen sich verschiedene Anwendungsgebiete, z.B. in der Bildverarbeitung oder im Mobilfunk [61], erkennen. Zum Anderen besitzt die CPU zur Datenverarbeitung ein Rechenwerk für arithmetische und logische Instruktionen (engl.: *arithmetic logic unit*, ALU), in dem sämtliche Operationen, die unterstützt werden, zusammengefasst sind. Die jeweilige Operation sowie die ausgewählten Operanden z.B. Speicherressourcen, sind dabei in der Instruktion angegeben.

Diese Art der Signalverarbeitung hat zur Folge, dass bei Prozessoren keine direkte und sequenzielle Abbildung einzelner Terme und Operanden erkennbar ist, was den

Aufwand für die Konzeption des Prozessorentwurfs deutlich erhöht. Des Weiteren wirkt sich die Verwendung eines ALU-basierten Datenpfades nachteilig auf die Performance aus. Obwohl es möglich ist, z.B. mit speziellen Befehlssatzarchitekturen mehrere Operationen innerhalb einer Instruktion parallel zu verarbeiten (engl.: *very large instruction word*, VLIW), ist es praktisch unmöglich, die hohe Auslastung der festverdrahteten Schaltungen für den Datenpfad zu erreichen. Darüber hinaus müssen Prozessoren programmiert werden, d.h. die Abarbeitung des Algorithmus muss an die zur Verfügung stehenden Hardwareressourcen angepasst werden. Der Vorteil von Prozessoren ist die signifikante Erhöhung der Flexibilität verglichen mit der festverdrahteten Architektur, womit nachträgliche Abänderungen des Algorithmus möglich sind. Zudem erlauben gängige Entwurfstools und -sprachen, z.B. der *Processor Designer* und LISA [87], einen Hardwareentwurf mit vergleichsweise kurzer Entwurfszeit. Der prinzipielle Aufbau von Prozessoren ist in Abb. 2.3 gegeben.

Koprozessoren Um den Datendurchsatz bestehender Schaltungen zu steigern, kann neben den vorangehend angesprochenen festverdrahteten Schaltungen auch auf programmierbare Hardwarebeschleuniger, sogenannte Koprozessoren, zurückgegriffen werden, deren grundlegender Aufbau dem eines ASIP ähnlich ist [55]. So findet auch hier eine instruktionsbasierte Zuweisung von Operanden des internen Datenpfads zur Signalverarbeitung statt. Das Einsatzgebiet ist i.d.R. auf spezielle Anwendungen beschränkt, z.B. die Berechnung kryptographischer Funktionen [80]. Im Betrieb wird der Koprozessor von einer übergeordneten CPU angesteuert. Eine Möglichkeit der Umsetzung stellt die Installation des Koprozessors als Prozessorperipherie dar. Ferner ist eine Anbindung direkt an die CPU möglich, was einerseits die Betriebsgeschwindigkeit erhöht, andererseits aber mit vergleichsweise hohem Entwurfsaufwand einhergeht. I.d.R. werden Koprozessoren verwendet, um die Performance, zumeist den Datendurchsatz oder Energieverbbrauch, existierender Hardwareentwürfe zu erhöhen.

Rekonfigurierbare Architekturen Die grundlegende Idee rekonfigurierbarer Hardwarearchitekturen basiert auf der Verwendung rekonfigurierbarer Hardwaremodule, auf die ein gewünschtes Schaltungsverhalten abgebildet werden kann. Neben etablierten Ansätzen, die eine solche Rekonfiguration nur initial zulassen, ist bei heutigen Ansätzen auch eine Änderung des Signalverarbeitungsverhaltens zur Laufzeit möglich [74]: Steht eine entsprechende Signalverarbeitungsaufgabe an, wird der Datenpfad zunächst entsprechend konfiguriert, danach kann dieser zur Signalverarbeitung verwendet werden. Die Granularität der rekonfigurierbaren Hardwaremodule kann dabei je nach Anwendung stark variieren.

Einen Spezialfall rekonfigurierbarer Architekturen stellen Prozessoren mit erweiterbarem Befehlssatz dar, die als eine Prozessorschaltung mit variablem Instruktionssatz angesehen werden können [55]. Dabei kommen neben dem Standard-Befehlssatz rekon-

figurierbare Hardwareelemente vor, die dynamisch weitere Befehle erstellen können. Eine Besonderheit ist, dass die Anpassung des Befehlssatzes an die Datenverarbeitung automatisch geschehen kann. In einem sogenannten *Profiling*-Schritt findet eine Analyse des abzuarbeitenden Programms statt. Aus den gewonnenen Informationen, z.B. häufig aufeinanderfolgende Operationen, können neue Instruktionen generiert werden, die durch Verwendung der rekonfigurierbaren Hardwaremodule dem Prozessor zur Verfügung gestellt werden.

Obgleich rekonfigurierbare Architekturen sowohl hohe Performance als auch hohe Flexibilität erzielen, ist der Programmierungs-, Implementierungs- und Entwurfsaufwand vergleichsweise hoch [91]. Auch geht die Konfiguration des Datenpfades bzw. des Instruktionssatzes zumeist mit hohem Zeitaufwand einher, weswegen rekonfigurierbare Architekturen zumeist für aufwändige Anwendungen, die sich z.B. aus mehreren Unteranwendungen zusammensetzen können, eingesetzt werden.

Multiprozessor-Systeme Der Einsatz von Multiprozessor-Systemen (engl.: *multi processor system-on-chip*, MPSoC), speziell im Bereich der Mehrzweck-Hardwaresysteme, hat in letzter Zeit deutlich an Bedeutung gewonnen [42]. Aufgrund der physikalischen Grenzen, an die einzelne Prozessoreinheiten bezüglich Taktfrequenz und Strukturgröße stoßen, und der ungebrochen steigenden Nachfrage nach höherer Performance ist die Verteilung der Rechenlast auf mehrere Prozessorkerne ein vielbeachteter Ansatz. Im Hinblick auf die Erhöhung der Performance für MPSoCs steht daher nicht mehr die Verarbeitung einzelner Prozessoreinheiten im Vordergrund, sondern die Verteilung der Aufgaben und die zugehörige Datenübertragung. Neben der eigentlichen Hardwarearchitektur der einzelnen Prozessorelemente muss zudem eine optimale Verbindungstopologie oder Hierarchie der Prozessoren gewählt werden, wodurch sich der Entwurfsaufwand deutlich erhöht. Da im Gegensatz zu rekonfigurierbaren Systemen auf bereits bestehende Prozessorkerne zurückgegriffen werden kann, ist von einer geringeren Entwurfszeit auszugehen, was jedoch zu Lasten der Effizienz geht [91]. MPSoC-Systeme werden daher i.d.R. zur Realisierung sehr hoher Datendurchsätze eingesetzt [89].

2.3 Implementierungstechniken

Ein wichtiger Aspekt für die ASIC-basierte Signalverarbeitung ist die Art der Hardwareimplementierung. So lassen sich verschiedene sogenannte Implementierungstechniken erkennen, durch deren Anwendung die Performance von Hardwareentwürfen variiert werden kann, das funktionale Verhalten sich jedoch nicht verändert [55]. Eine Übersicht gängiger Implementierungstechniken ist im Folgenden sowie graphisch in Abb. 2.4 und Abb. 2.5 gegeben.

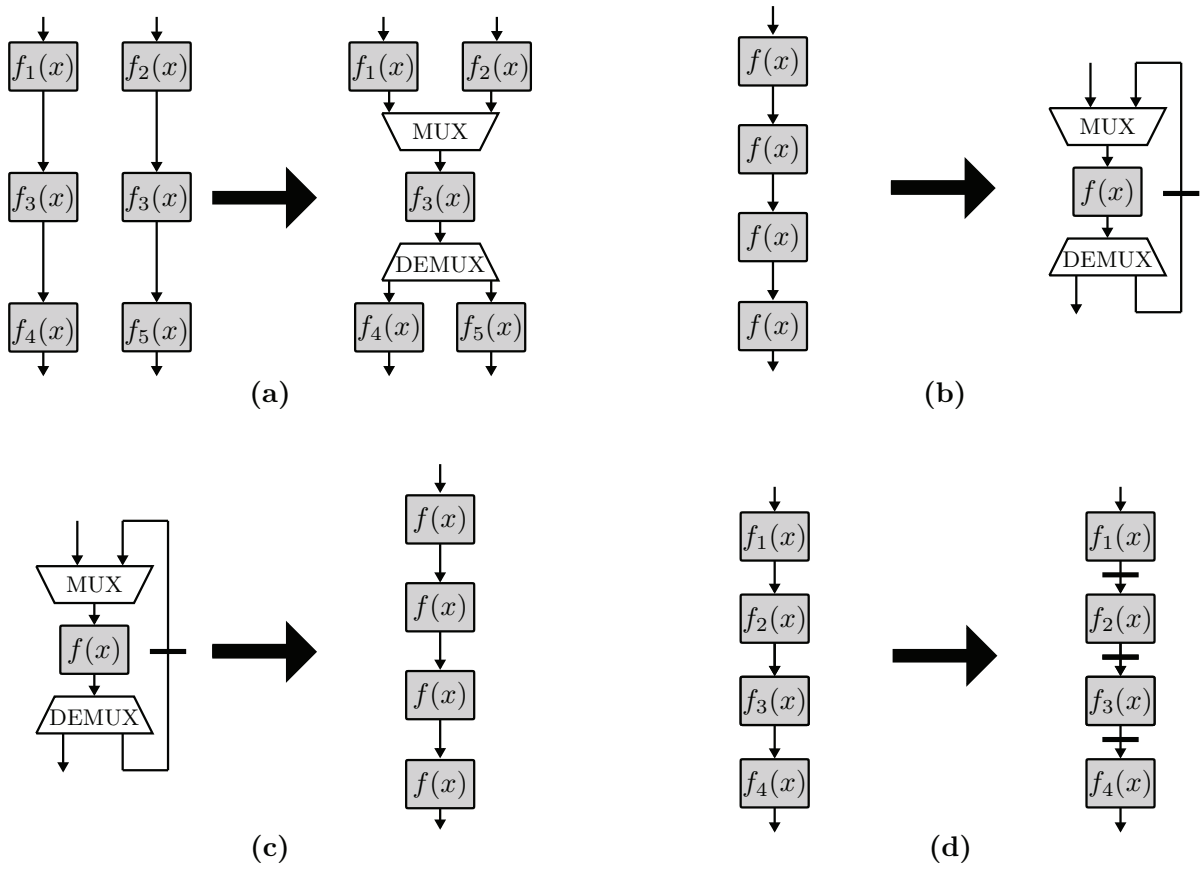


Abbildung 2.4: Beispielhafte Anwendung der Implementierungstechniken **(a)** Zeitmultiplex, **(b)** iterative Zerlegung, **(c)** *Loop-Unrolling* und **(d)** *Pipelining* mit $f_i(x)$ als beliebige, kombinatorische Funktion.

Zeitmultiplex Zeitmultiplex (engl.: *multiplexing / time-sharing*) stellt eine Methode zur Mehrfachnutzung einzelner Hardwareressourcen dar [55]. Hierbei werden gleiche Operatoren einer Schaltung zu einem einzelnen Operator zusammengefasst und die Verdrahtung des Datenpfades durch Multiplexer am Eingang und Demultiplexer am Ausgang erweitert (siehe Abb. 2.4a). Diese Implementierungstechnik wird i.d.R. dazu eingesetzt, mehrfach auftretende algorithmische Operatoren oder kombinatorische Elemente zusammenzufassen und somit die Gesamtgröße der Schaltung zu reduzieren. Da der Einsatz von Multiplexern und Demultiplexern mit zusätzlicher Steuerlogik zur Auswahl des aktuellen Datenpfades einhergehen muss, ist die Anwendung der Zeitmutiplex-Technik für entsprechend große Hardwaremodule sinnvoll. Des Weiteren muss sichergestellt sein, dass keine Mehrfachbeanspruchung der Ressource vorliegt. Ein solcher Datenkonflikt kann zum Einen durch entsprechende Berücksichtigung dieses Kriteriums bei der Auswahl gelöst werden, indem der Zeitmultiplex nur für Elemente zulässig ist, die nicht im selben Takt ausgeführt werden. Zum Anderen kann die Verarbeitung durch das Einfügen zusätzlicher Takte realisiert werden, was sich negativ auf die Latenz auswirkt.

Iterative Zerlegung Ein Spezialfall des Zeitmultiplex ist die iterative Zerlegung, die bei Schaltungen angewendet wird, die aus mehreren Abarbeitungsschritten gleichen Aufbaus bestehen [69]. Auch hierbei wird die Schaltungskomplexität reduziert, indem diese Hardwareelemente zusammengefasst werden (siehe Abb. 2.4b). Zusätzlich wird der Hardwareentwurf mit einer registerbasierten Rückführung versehen, die eine taktbasierte, iterative Abarbeitung erlaubt.

Loop-Unrolling Das Grundprinzip des *Loop-Unrolling* kann als sequenzielle Umkehrung der vorangehend vorgestellten iterativen Zerlegung angesehen werden, die bei iterativer Datenverarbeitung angewendet wird [69]. Aus algorithmischer Sicht lässt sich das *Loop-Unrolling* als Aufbrechen einer Schleife verstehen (siehe Abb. 2.4c). Daher ist diesem Verfahren speziell im Bereich der iterativen näherungsweisen Bestimmung von Ergebnissen besondere Bedeutung beizumessen, da durch Kombination mit der *Pipelining*-Implementierungstechnik die Reduktion des kritischen Pfades und die Erhöhung des Datendurchsatzes erzielt werden kann. Nachteilig wirkt sich die Anwendung auf die Schaltungskomplexität aus, da die Vervielfachung von Schaltungsteilen höheren Platzbedarf erfordert.

Pipelining *Pipelining* lässt sich grundsätzlich als Aufteilung eines vollständig kombinatorischen Daten- oder Steuerpfads in mehrere Unterpfade beschreiben [44]. Hierzu werden sogenannte *Pipelining*-Register eingefügt und somit der kritische Pfad minimiert, was zu einer Steigerung der Taktfrequenz führt (siehe Abb. 2.4d). Bei mehrfacher Nutzung einzelner Elemente im Pfad, z.B. Speichermodule als Datenquelle und -senke in Prozessorschaltungen, muss zusätzlich das Auftreten von Daten- und Steuerkonflikten unterbunden werden. Obgleich *Pipelining* sich positiv auf die Taktfrequenz auswirkt, verschlechtert sich die Latenz, also die taktbasierte Verzögerung, je nach Anzahl der zusätzlichen Registerstufen. Der Energieverbrauch hingegen verringert sich i.d.R., was mit geringeren Signaltreibern am Ausgang der Gatter zu erklären ist.

Parallelisierung Die Parallelisierung kann, wie das *Loop-Unrolling*, auch als Aufbrechen der Hardwareumsetzung einer Schleife, also einer Struktur, die eine feste Abarbeitungssequenz mehrmals durchläuft, verstanden werden (siehe Abb. 2.5a) [55]. Im Gegensatz zum *Loop-Unrolling* darf hierbei jedoch keine Abhängigkeit zum im vorangehenden Schritt berechneten Zwischenergebnis bestehen, d.h. die Datenverarbeitung innerhalb der Schleife muss unabhängig und somit parallel durchführbar sein. Parallelisierung hat somit die Verringerung der Latenz zur Folge; die höhere Komplexität bleibt hingegen bestehen.

Retiming Das Verfahren des *Retiming* folgt meist auf vorab durchgeführte signifikante Änderungen des Schaltungsaufbaus [55]. Ziel ist es, die Taktdauer unterschied-

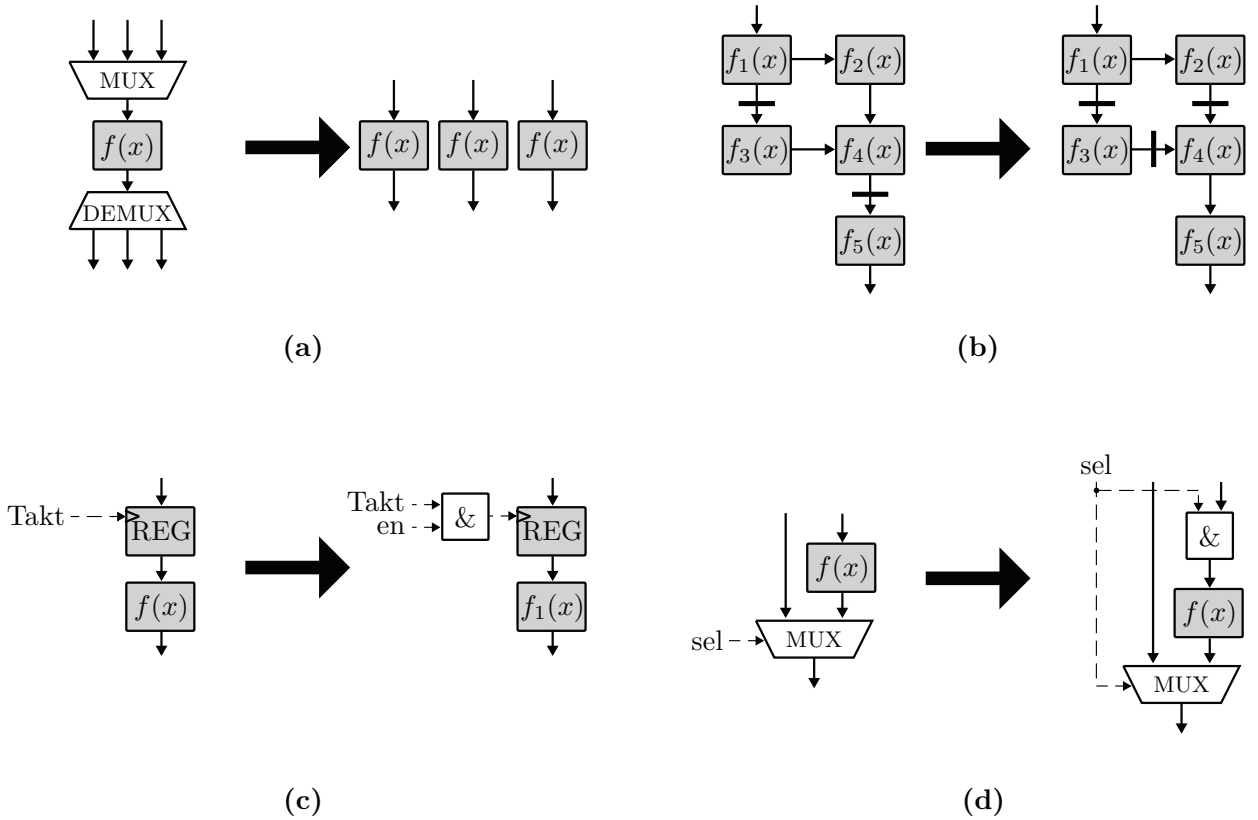


Abbildung 2.5: Beispielhafte Anwendung der Implementierungstechniken **(a)** Parallelisierung, **(b)** *Retiming*, **(c)** *Clock-Gating* und **(d)** *Operand-Isolation* mit $f_i(x)$ als beliebiger, kombinatorischer Funktionen. REG bezeichnet Registerbänke, en und sel sind die Steuersignale für *Clock-Gating* und *Operand-Isolation*.

licher Pfade innerhalb des Hardwareentwurfs möglichst gleichlang zu halten, was eine Effizienzsteigerung durch höhere parallele Auslastung der vorhandenen Ressourcen zur Folge hat (siehe Abb. 2.5b). Dabei ist auch das Aufbrechen komplexerer Funktionseinheiten oder Operanden, wie z.B. Multiplizierern, denkbar. Auch der Energieverbrauch kann durch geringere Treiberstärken an den Gatterausgängen reduziert werden.

Clock-Gating *Clock-Gating* ist eine Implementierungstechnik, die sich in erster Linie mit der Verringerung des Energieverbrauchs sequenzieller Schaltungselemente befasst [58]. Die grundlegende Idee basiert auf dem Abschalten ungenutzter Hardwaremodule einer Schaltung durch Deaktivierung des anliegenden Taktsignals mittels entsprechender Torschaltung (siehe Abb. 2.5c). Hierdurch werden die Umschaltvorgänge der nachfolgenden kombinatorischen Schaltungselemente unterbunden, was i.d.R. zu einer Steigerung der Energieeffizienz führt.

Operand-Isolation Die grundlegende Idee der *Operand-Isolation*-Technik ist die Reduktion des dynamischen Energieverbrauchs durch Abschalten nicht benötigter Operationen im Datenpfad [93]. Je nach Bedarf kann durch eine entsprechende Torschaltung ein Datenpfadabschnitt mit einem gewünschten Wert angesteuert werden (siehe Abb. 2.5d). Neben den festen Werten Null bzw. Eins durch OR- bzw. AND-Gatter, können zudem auch Latches verwendet werden, die die zuletzt angelegten Eingangsdaten zwischenspeichern. Für beide Ansätze sind zusätzliche Hardwareelemente notwendig, weswegen die Komplexität leicht ansteigt. Wird *Operand-Isolation* im kritischen Pfad angewendet, sinkt zudem auch die Taktfrequenz.

2.4 ASIC-Hardwareentwurf

Der Begriff Hardwareentwurf umfasst den Ablauf der Schaltungsentwicklung, ausgehend von der Spezifikation des gewünschten Verhaltens bis zum fertigen Chip [123]. Im Rahmen dieser Arbeit liegt der Fokus dabei auf der Angabe von Entwurfskriterien, der Spezifikation, dem logischen und physikalischen Entwurf und den zugehörigen Verifikationsschritten von ASICs, weswegen diese Aspekte nachfolgend näher erläutert werden.

2.4.1 Entwurfskriterien

Wichtiger Bestandteil des ASIC-Entwurfs ist die Formulierung von Entwurfskriterien (engl: *constraints*), die sich aus vorab spezifizierten Zielen oder Randbedingungen des Einsatzgebietes ergeben [123]. Es können dabei sowohl Anzahl als auch Priorität der Entwurfskriterien variieren. Zudem lassen sich neben technischen auch weitere Aspekte, wie z.B. ökonomische Faktoren, hierzu zählen. Allgemeine Entwurfskriterien des digitalen Schaltungsentwurfs sind u.a.

- Komplexität,
- Taktfrequenz bzw. -dauer,
- Latenz,
- Energieverbrauch,
- Rechengenauigkeit,
- Entwurfszeit und
- finanzielle Kosten.

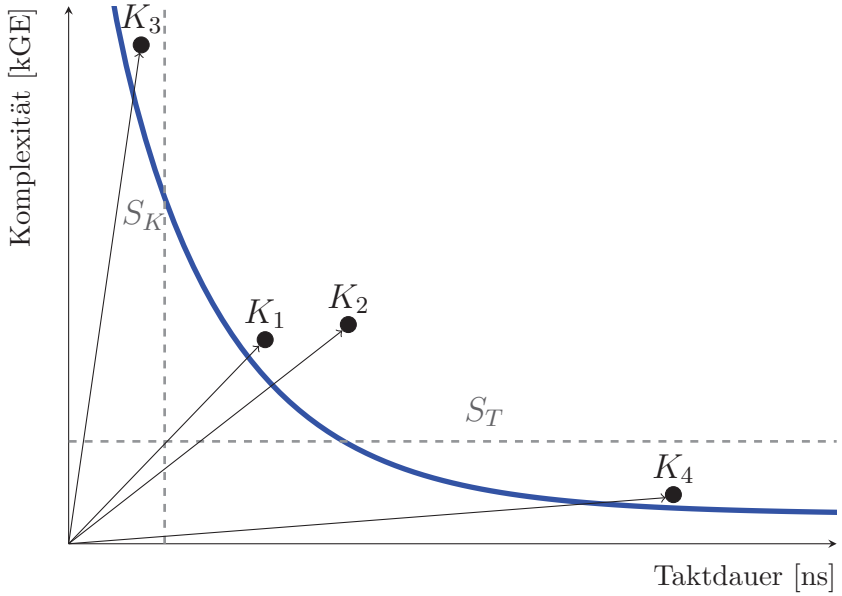


Abbildung 2.6: Zweidimensionaler Hardwareentwurfsraum mit dem Verlauf einer Pareto-Front (blaue Linie) für unterschiedliche HDL-Implementierungen gleicher Funktion. K_1 stellt einen Pareto-Punkt dar, der die nicht-optimale Konfiguration K_2 dominiert. K_3 bzw. K_4 entstehen z.B. durch Anwendung der Implementierungstechniken *Loop-Unrolling* bzw. iterative Zerlegung. S_K bzw. S_T stellt eine mögliche Spezifikation für die Komplexität bzw. Taktdauer dar.

Besondere Bedeutung kommt im Rahmen dieser Arbeit dem Entwurfskriterium Rechengenauigkeit zu, das daher in Kap. 3.2 ausführlich behandelt wird.

Mathematisch ausgedrückt kann die Performance jedes Hardwareentwurfs als Vektor eines m-dimensionalen Raums, des sogenannten Hardwareentwurfsraums, verstanden werden, wobei jedes Entwurfskriterium eine Dimension aufspannt [76]. Durch Abänderungen des Verhaltens kann der zugehörige Aufpunkt verschoben werden, wobei dies, z.B. bei Anwendung der in Kap. 2.3 angesprochenen Implementierungstechniken, nicht zwangsläufig mit einer Änderung der gewünschten Funktion einhergehen muss. Für eine gewünschte Schaltungsfunktionalität kann somit ein Performanceoptimum ermittelt werden. Da zumeist mehrere Kriterien gleichzeitig betrachtet werden müssen, wird in diesem Zusammenhang auch von multikriterieller oder Pareto-Optimierung gesprochen. Entwurfskriterien, die nicht weiter verbessert werden können, ohne ein oder mehrere andere Kriterien zu verschlechtern, werden Pareto-Punkte genannt. Die Menge aller Pareto-Punkte bildet die sogenannte Pareto-Front des Hardwareentwurfraums [76]. Ziel des Hardwareentwurfs ist es, Aufpunkte im Hardwareentwurfsraum zu bestimmen, die sämtliche vorangehend spezifizierten Performanceanforderungen einhalten. Ein Beispiel für die Bewertung verschiedener Entwürfe im Hardwareentwurfsraum ist in Abb. 2.6 gegeben.

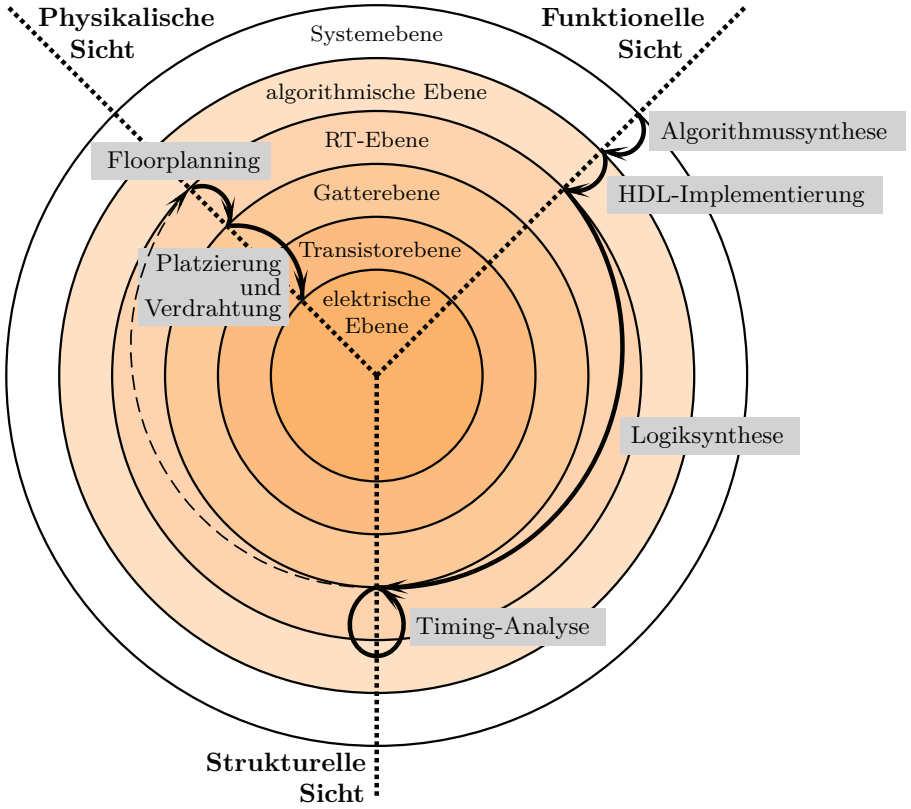


Abbildung 2.7: Erweitertes Gajski-Diagramm nach [121] mit typischem Ablauf des ASIC-Entwurfs ohne Verifikations- oder Validierungsschritte. Die Ringe bzw. die gepunkteten Linien markieren die Entwurfsebenen bzw. Entwurfsdomänen. Durchgezogene Pfeile entsprechen den grau unterlegten Entwurfsschritten.

2.4.2 Entwurfsschritte

Der ASIC-Hardwareentwurf besteht aus mehreren Entwurfsschritten, die unterschiedlichen Entwurfsebenen zugeordnet werden können [123]. Einen graphischen Überblick hierfür liefert das Gajski-Diagramm, oder auch Y-Diagramm, das 1983 von Daniel Gajski und Robert Kuhn erstmals erwähnt [32] und 1985 von Robert Walter und Donald Thomas weiter entwickelt [121] wurde. Es ermöglicht eine Betrachtung von Hardwareentwürfen auf verschiedenen Abstraktionsebenen und durch verschiedene Entwurfssichten. Ein allgemeiner Überblick über das Gajski-Diagramm sowie der gängige Ablauf des standardzellenbasierten ASIC-Entwurfs ist in Abb. 2.7 gegeben.

Die Spezifikation stellt den Anfang des Hardwareentwurfs dar. Das gewünschte Verhalten der entstehenden Schaltung wird dabei hinreichend exakt definiert. Je nach Anforderung an das gewünschte Verhalten werden zudem i.d.R. Vorgaben an unterschiedliche Entwurfskriterien formuliert (siehe Abb. 2.6). Die Einhaltung beider Aspekte auf jeder Entwurfsebene ist das übergeordnete Ziel des Hardwareentwurfs. Die Spezifika-

tion kann zudem in vielen verschiedenen Beschreibungsformen, z.B. in Textform oder als Blockdiagramm, angegeben werden.

Ausgehend von der Spezifikation muss die Schaltung in eine HDL-Beschreibung übersetzt werden: Je nach Anwendung muss hierfür ggf. vorab ein Algorithmus aus der Spezifikation extrahiert werden, der die geforderte Funktionalität erfüllt. Da der Fokus dieser Arbeit auf der effizienten Umsetzung elementarer Funktionen liegt, werden bestehende Algorithmen als Spezifikation verwendet, wodurch dieser Schritt vernachlässigt werden kann.

Bei der HDL-Implementierung wird der Hardwareentwurf auf Register-Transfer-Ebene (RT-Ebene) überführt, auf der z.B. synchrone Elemente der Schaltung bereits taktgenau angegeben werden, wodurch die Verwendung von sequenziellen Elementen wie Registern notwendig ist. Zudem wird der Datenpfad durch gesteuerte Rechenwerke realisiert. Kombinatorische Schaltungselemente sind zumeist als funktionelle Blöcke beschrieben. Für die HDL-Implementierung von Algorithmen existiert eine Vielzahl unterschiedlicher Ansätze. Das klassische Vorgehen basiert auf der manuellen Umsetzung mittels Hardwareentwurfssprachen wie VHDL oder Verilog [50] [51]. Obgleich diese Vorgehensweise i.d.R. mit hoher Performance einhergeht, kann dies für komplexe Algorithmen mit unpraktikabel hoher Entwurfszeit einhergehen. Es gibt daher unterschiedliche Programme und Hochsprachen, die eine *High-Level*-Synthese (HLS) zur automatisierten HDL-Implementierung ermöglichen. Einen Ansatz für effizienten ASIP-Entwurf stellt der *Processor Designer* von Synopsys dar. Die zugrunde liegende Programmiersprache LISA besitzt speziell angepasste Datentypen, Operatoren, Ausdrücke und Anweisungen zur Definition des gewünschten Hardwareverhaltens. Neben der HLS lassen sich hiermit auch Compiler bzw. C-Modelle zur Programmierung bzw. Modellierung erstellen.

Nach der Entwurfseingabe wird das vorhandene Hardwaremodell in mehreren Schritten in detailliertere Beschreibungen übersetzt, bis am Ende die Beschreibung der Schaltung physikalisch als Layout vorliegt. Wichtige Syntheseschritte sind die Logiksynthese, bei der die HDL-Implementierung in eine Gatternetzliste übersetzt wird, und die Layoutsynthese, die u.a. die Schaltungselemente auf der Chipfläche platziert und verdrahtet. Allgemein wird die Synthese eines Hardwareentwurfs anhand bestehender Entwurfskriterien durchgeführt, wozu in späteren Entwurfsphasen auch prozessbedingte Kriterien hinzukommen. I.d.R. werden Syntheseschritte zudem rechnergestützt durchgeführt.

Neben der Schaltungssynthese sind Verifikation und Validierung wichtige Schritte des Hardwareentwurfs, die zum Auffinden von Entwurfsfehlern eingesetzt werden. Sie stellen üblicherweise sehr zeitintensive Aspekte des Hardwareentwurfs dar [40]. Dabei kommen je nach Entwurfsebene unterschiedliche Verfahren, wie Verhaltenssimulation, Analyse des Zeitverhaltens oder formale mathematische Beweisführungen, zum Einsatz. Allgemein können Entwurfsfehler verschiedene Ursachen, wie z.B. unzureichen-

de Spezifikation, Implementierungsfehler oder Verletzung der Entwurfsregeln, haben [40]. Während die Validierung experimentellen Charakter aufweist, d.h. die vollständige Richtigkeit des untersuchten Aspekts nicht nachgewiesen wird, basiert die Verifikation stets auf einer formalen Beweisführung, womit Entwurfsfehler ausgeschlossen werden können. Für den ASIC-Entwurf hat sich der Einsatz von Verhaltenssimulationen bewährt [67]: Hierbei werden der aktuelle Hardwareentwurf mit gängigen Eingangsdaten bedient und die Ergebnisse mit den Vorgaben der Spezifikation verglichen [31]. Im Rahmen dieser Arbeit werden Hardwareentwürfe unter Zuhilfenahme eines Modells sowohl auf algorithmischer Ebene, als auch auf RT- und Gatterebene validiert. Zudem werden Timing-Analysen nach der Logik- und physikalischen Synthese sowie ein *Design-Rule-Check* (DRC) auf Transistorebene standardmäßig durchgeführt [20].

2.5 Zusammenfassung

In diesem Kapitel sind Grundlagen des Aufbaus und Hardwareentwurfs von ASICs beschrieben. Die Angabe der Hardwarearchitekturen und Implementierungstechniken bietet eine Übersicht gängiger Ansätze zur Umsetzung von Algorithmen. Ausgehend von den Beschreibungen zum ASIC-Hardwareentwurf kann auf die hardwarebasierte Berechnung elementarer Funktionen und die damit einhergehenden Herausforderungen eingegangen werden, was die Einführung des im Rahmen dieser Dissertation verwendeten HLS-Werkzeugs zur automatischen Funktionsapproximation erlaubt (siehe Kap. 3.7). Beide Aspekte werden im folgenden Kapitel betrachtet.

3 Digitale Signalverarbeitung elementarer Funktionen

Digitale Signalverarbeitung bezeichnet allgemein die Modifikation zeit- und wertediskreter Signale, was u.a. bei der Berechnung von Algorithmen durch Hardwarestrukturen notwendig ist [90]. Hierfür sind sowohl die Darstellung von Zahlen als auch die Verarbeitung elementarer Funktionen von essenzieller Bedeutung, weswegen beide Aspekte in diesem Kapitel vorgestellt werden. Im Anschluss (Kap. 3.7) wird das im Rahmen der Dissertation verwendete Verfahren der automatischen Funktionsapproximation eingehend beschrieben, das die Grundlage für die in Kap. 4 und Kap. 5 behandelten Implementierungen bildet.

3.1 Digitale Zahlendarstellung

Auswahl und Darstellungsweise des verwendeten Zahlensystems digitaler Schaltungen haben, u.a. wegen vielseitiger Realisierungsmöglichkeiten des betrachteten Algorithmus, direkten Einfluss auf die Performance der zugehörigen Schaltung. Aufgrund der Beschaffenheit digitaler Systeme stellt das Binärsystem (oder auch Dualsystem) das bestgeeignete Zahlensystem zur digitalen Signalverarbeitung dar. Allgemein besitzt jedes Zahlensystem eine feste Ziffernmenge

$$z \in \{0, 1, 2, \ldots, r-1\} \ , \tag{3.1}$$

mit r als Radix des Zahlensystems (siehe Tab. 3.1). Im Binärsystem ($r = 2$) werden Ziffern als Bit (engl.: *binary digit*) bezeichnet. Durch Gruppierung mehrerer Bits lässt sich der Zahlenraum erweitern, da jedem so entstehenden Bitvektor ein Zahlenwert zugeordnet werden kann [84]. Diese Zuweisung wird als Zahlendarstellung oder Zahlenformat bezeichnet. Je nach Art der Zahlendarstellung variieren sowohl der Umfang des Wertebereichs als auch die Komplexität arithmetischer Operationen und die Performance der digitalen Signalverarbeitung. Allgemein lassen sich zwei Ansätze der binären Zahlendarstellung unterscheiden: Festkomma- und Gleitkomma-Zahlenformate. Eine Sonderform stellt das logarithmische Zahlenformat dar, welches als Kombinati-

Tabelle 3.1: Überblick über gängige Zahlensysteme der Computerarithmetik nach [116].

Zahlensystem	Radix r	Ziffernmenge Z
Binärsystem	2	{ 0,1 }
Oktalsystem	8	{ 0,1,2,3,4,5,6,7 }
Dezimalsystem	10	{ 0,1,2,3,4,5,6,7,8,9 }
Hexadezimalsystem	16	{ 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F }

on beider allgemeinen Ansätze angesehen werden kann (siehe Kap. 3.1.3). Alle drei Zahlenformate werden im Folgenden vorgestellt.

3.1.1 Festkomma-Zahlenformat

Bei Festkommazahlen ist das Komma fest zwischen zwei benachbarten Bits angegeben. Zudem liegt i.d.R. eine stellenwertbasierte Zahlendarstellung nach [10] gemäß

$$V(Z_F) = \sum_{i=-m}^{n} z_i r^i \ ; \ m \in \mathbb{N}, \ n \in \mathbb{N}^+, \ z_i \in Z_F \ , \tag{3.2}$$

mit r als Radix und Z_F als Festkomma-Ziffernmenge, vor. Für das Binärsystem, welches im Folgenden ausschließlich betrachtet werden soll, ergibt sich unter Anpassung des Index n bzw. m an die Bitlänge des ganzzahligen bzw. gebrochenen Anteils zu

$$V(Z_F) = \sum_{i=-m}^{n-1} z_i 2^i; \ m \in \mathbb{N}, \ n \in \mathbb{N}^+, \ z_i \in \{0,1\} \ . \tag{3.3}$$

Aus diesem elementaren Ansatz können viele unterschiedliche festkommabasierte Zahlendarstellungen abgeleitet werden. Gängige Beispiele hierfür sind im Folgenden beschrieben sowie in Abb. 3.1 visualisiert.

Integer *Integer* sind eine einfache Möglichkeit der festkommabasierten Zahlendarstellung. Ausgehend von Gl. (3.3) ist jeder Ziffer eines Bitvektors ein entsprechender Stellenwert zugeordnet. *Integer* besitzen keine Nachkommastellen ($m = 0$). Darüber hinaus ist eine Unterteilung in natürliche bzw. vorzeichenbehaftete Zahlen (engl.: *Unsigned Integer* bzw. *Signed Integer*) üblich. Es ergibt sich für natürliche Zahlen

$$Z_{uint} \ = \ z_{n-1} \ \ z_{n-2} \ \ \ldots \ \ z_1 \ \ z_0 \ . \tag{3.4}$$

Es wird somit ein Zahlenbereich von $0 \leq V(Z_{uint}) \leq 2^n - 1$ abgedeckt.

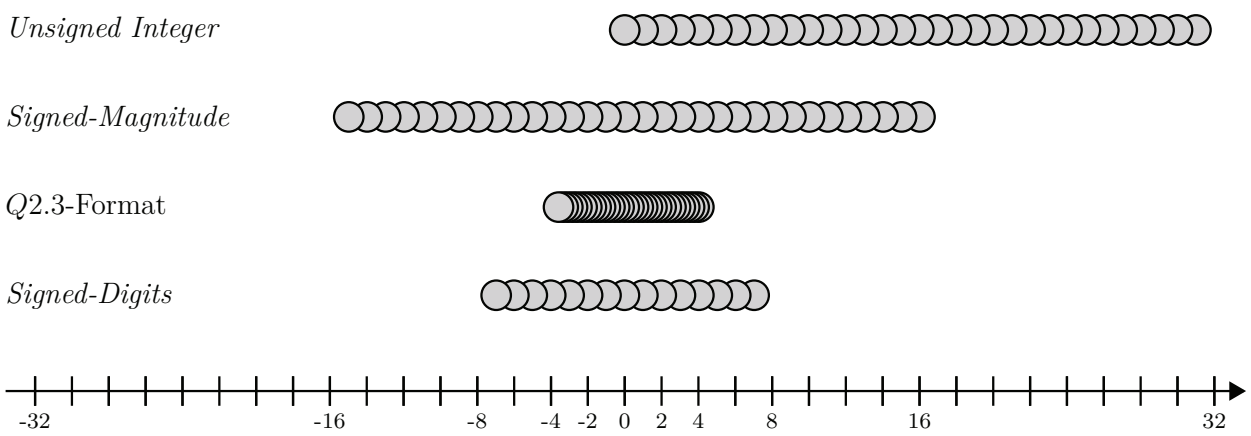


Abbildung 3.1: Zahlenbereiche der in Kap. 3.1.1 beschriebenen Festkomma-Zahlenformate nach [84] für eine 6 Bit breite Binärzahl. Soweit nicht anders spezifiziert ist die Darstellung ganzzahlig und im Zweierkomplement.

Für negative Zahlen existieren mehrere Möglichkeiten der Umsetzung. Ein Ansatz ist die Bildung des Komplements, d.h. also der Negierung jeder Ziffer. Die Verwendung des Zweierkomplements besitzt für negative Zahlen die Darstellungsvorschrift

$$\overline{Z}_{int} \;=\; 1 \;\; \overline{z}_{n-2} \;\; \overline{z}_{n-3} \;\; \ldots \;\; \overline{z}_1 \;\; \overline{z}_0 \;\; +1 \;, \tag{3.5}$$

mit $\overline{z}_i$ als negierte Ziffer an der i-ten Stelle und $\overline{Z}_{int}$ als negative, ganze Zahl. Das höchstwertige Bit $v = z_n$ fungiert als Vorzeichenbit, das durch $v = 0$ bzw. $v = 1$ positive bzw. negative Zahl kennzeichnet. Somit besitzt der ganzzahlige Anteil eine Bitlänge von $n - 1$. Der Wertebereich umfasst daher

$$-2^{n-1} \leq V(Z_{int}) \leq 2^{n-1} - 1 \;. \tag{3.6}$$

Durch die endliche Größe des Wertebereichs kann es bei arithmetischen Operationen zu Überläufen (engl.: *overflow*) kommen, die ggf. durch zusätzliche Logik in den Signalverarbeitungseinheiten erkannt und kompensiert werden müssen.

Q-Zahlenformat Eine Möglichkeit der Darstellung gebrochener Zahlen bietet das Q-Zahlenformat (auch Q-Format genannt) [8]. Es besteht aus einem ganzzahligen und einem gebrochenen Anteil

$$Z_Q \;=\; z_{n-1} \;\; z_{n-2} \;\; \ldots \;\; z_1 \;\; z_0 \;\; , \;\; z_{-1} \;\; z_{-2} \;\; \ldots \;\; z_{-m+1} \;\; z_{-m} \tag{3.7}$$

und wird durch $Qn.m$, für $n = 0$ vereinfacht durch $Q.m$, beschrieben. Die Kommastelle ist dabei nicht explizit vorgegeben, sondern muss bei der Durchführung mathematischer

Operatoren manuell berücksichtigt werden. Zur Darstellung negativer Zahlen kann auf die bereits angesprochene Komplementbildung zurückgegriffen werden. Mit dieser Vorgehensweise kann das Q-Zahlenformat an Ganzzahl-Rechenwerke angepasst werden, die z.B. in Mikrocontrollern Verwendung finden. Durch Schiebeoperatoren kann zudem der Wertebereich verschoben werden, womit die vorangehend angesprochenen Überläufe vermieden werden können. Das Q-Zahlenformat stellt somit ein effizientes Zahlenformat zur digitalen Signalverarbeitung dar, das sowohl zur Implementierung des Datenpfads als auch zur Programmierung von z.B. Prozessoren eingesetzt werden kann.

Signed-Magnitude Ein weiterer Ansatz zur Darstellung negativer Zahlen ist durch direkte Angabe des Vorzeichens in der Ziffernmenge Z_{SM} möglich [63]. Dieses sogenannte *Signed-Magnitude* Verfahren kennzeichnet negative Zahlen wie das *Integer*-Zahlenformat durch das Vorzeichenbit v

$$\overline{Z}_{SM} \;=\; 1 \;\; z_{n-2} \;\; z_{n-3} \;\; \ldots \;\; z_1 \;\; z_0 \; . \tag{3.8}$$

Die Berechnung des zugehörigen Zahlenwerts verändert sich hingegen zu

$$V(Z_{SM}) = (-1)^v \sum_{i=0}^{n-2} z_i r^i \; . \tag{3.9}$$

Obgleich diese Notation eine einfache Form der Darstellung negativer Zahlen erlaubt, erweist sie sich bei einer Vielzahl arithmetischer Operationen als nachteilig. So ist bei unterschiedlichen Vorzeichen zweier Summanden zusätzlicher Verarbeitungsaufwand notwendig, der sich negativ auf die Performance auswirkt.

Signed-Digits Neben der Verwendung vorzeichenbehafteter Zahlen ist auch die Behandlung vorzeichenbehafteter Ziffern (engl.: *Signed-Digits*) eine Möglichkeit zur Zahlendarstellung. Der Vorteil liegt in der optimierten Zahlendarstellung hinsichtlich Ziffern, die einen Wert ungleich Null besitzen. Das Ziffernalphabet muss zudem um negative Ziffern erweitert werden. Im Binärsystem ergibt sich

$$z_i \in \{0, 1, -1\} \; . \tag{3.10}$$

Für die Umsetzung durch Elemente integrierter Schaltungen erhöht sich der Darstellungsaufwand, da 3^n mögliche Zahlenkombinationen einer n Bit langen Zahl existieren. Aus Gl. (3.2) ergeben sich allerdings nur $2^n - 1$ unterschiedliche Zahlenwerte, womit eine höhere Redundanz gegenüber den vorangehend benannten Verfahren erkennbar ist.

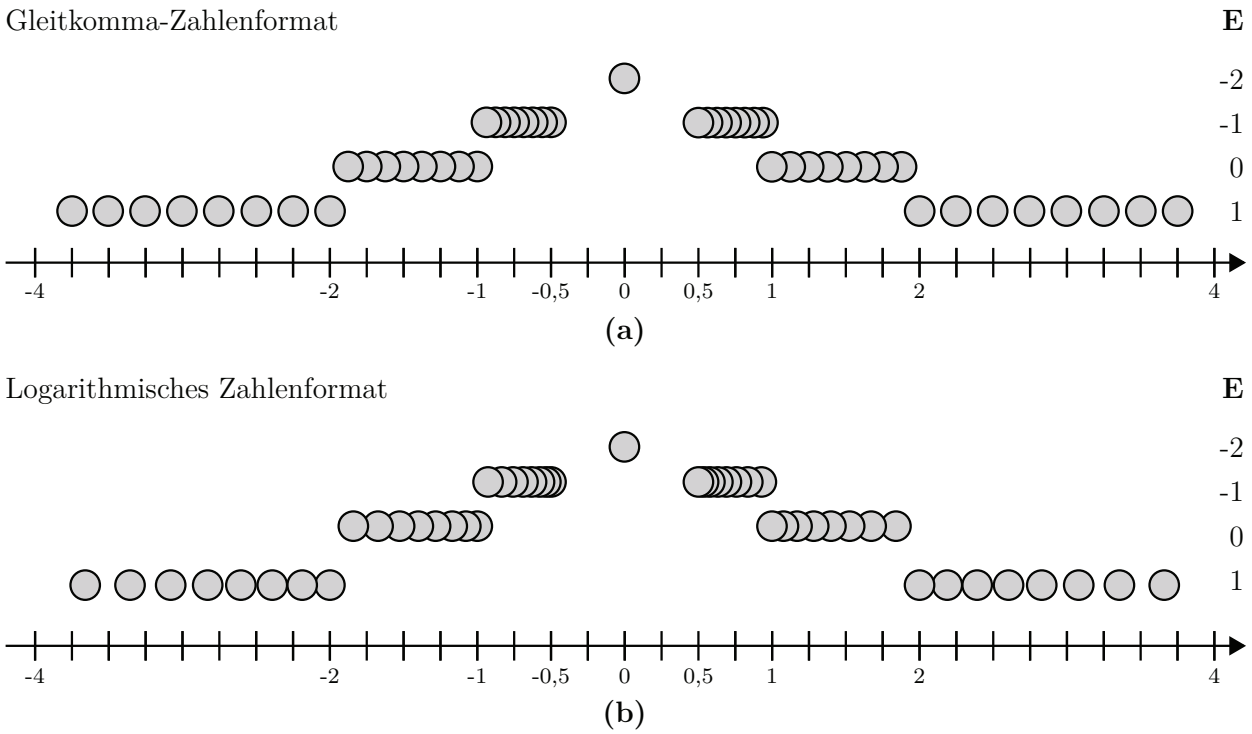


Abbildung 3.2: Zahlenbereiche **(a)** der Gleitkomma- und **(b)** der logarithmischen Zahlendarstellung nach [84] für eine 6 Bit breite Zahl mit $n_E = 2$ und $n_M = 3$ (siehe Kap. 3.1.2 und Kap. 3.1.3). Der Wert Null ist durch $E = -2$ festgelegt.

3.1.2 Gleitkomma-Zahlenformat

Gleitkommazahlen sind durch Angabe einer Mantisse und eines Exponenten definiert [35]. Sie werden zumeist mit hohen Bitbreiten realisiert, um Zahlenwerte möglichst genau darzustellen. Die binäre Zahlendarstellung lautet

$$Z_G = v \ \underbrace{z_{n_E} \ z_{n_E-1} \ \dots \ z_{m_E}}_{E_G} \ \underbrace{z_{n_M} \ z_{n_M-1} \ \dots \ z_0}_{M_G} \ , \tag{3.11}$$

mit dem Exponenten E_G, der normierten Mantisse M_G und dem Vorzeichenbit v. Eine übliche, normalisierte Rechenvorschrift des zugehörigen Zahlenwerts ist durch den in [52] festgelegten IEEE Standard als

$$V(Z_G) = (-1)^v \cdot 2^{E_G} \cdot (M_G + 1) \ , \tag{3.12}$$

mit

$$E_G = \pm \sum_{j=m_E}^{n_E} z_j 2^{(j-m_E)} \ \text{ und } \ M_G = \sum_{i=0}^{n_M} z_i 2^{(i-n_M-1)} \ , \tag{3.13}$$

definiert. Diese Art der Zahlendarstellung umfasst zwar einen hohen Wertebereich, benötigt jedoch auch höheren Aufwand zur Umsetzung einfacher arithmetischer Operatoren. Die Darstellung von Sonderfällen, z.B. $V(Z_G) = 0$, ist durch reservierte Werte des Exponenten möglich. Ein Überblick hierfür findet sich in [52], ein graphisches Beispiel ist in Abb. 3.2a gegeben.

3.1.3 Logarithmisches Zahlenformat

Das grundlegende Prinzip des logarithmischen Zahlenformats beruht auf der vollständigen Darstellung eines Zahlenwerts durch ein logarithmisches Äquivalent [79] [24]. Es kann als Mischform der vorangehend vorgestellten Festkomma- und Gleitkomma-Zahlenformate interpretiert werden, da es die Eigenschaften beider Ansätze kombiniert: So wird zum Einen eine stellenwertbasierte Zahlendarstellung verwendet (siehe Kap. 3.1.1). Zum Anderen wird im logarithmischen Zahlenformat wie im Gleitkomma-Zahlenformat das Komma durch Angabe des Exponenten verschoben, was einen entsprechend hohen Wertebereich zur Folge hat (siehe Kap. 3.1.2).

Für die Zahlendarstellung ergibt sich

$$Z_L \;=\; v \;\; \underbrace{z_{n_E} \;\; z_{n_E-1} \cdots \;\; z_{m_E}}_{E_L} \;\; \underbrace{z_{n_M} \;\; z_{n_M-1} \;\; \cdots \;\; z_0}_{M_L} \,, \tag{3.14}$$

mit v als Vorzeichen sowie der entsprechenden Gleitkomma-Interpretation des Vorzeichens v und des Exponenten E_L. Die Mantisse M_L liegt als normalisierter, logarithmischer Wert vor ($0 \leq M_L < 1$); die Berechnung des Zahlenwerts ist durch

$$V(Z_L) = (-1)^v \cdot 2^{E_L} \cdot 2^{M_L} \,, \tag{3.15}$$

mit

$$E_L = \pm \sum_{j=m_E}^{n_E} z_j 2^{(j-m_E)} \;\; \text{und} \;\; M_L = \sum_{i=0}^{n_M} z_i 2^{(i-n_M-1)} \,, \tag{3.16}$$

festgelegt (siehe Abb. 3.2b). Genau wie in Kap 3.1.2 können Sonderfälle z.B. über spezielle Werte des Exponenten angegeben werden.

Bei dieser Art der Zahlendarstellung werden Zahlen also durch ein Zweierpotenz-Äquivalent formuliert, was für bestimmte Operationen, z.B. Multiplikationen und Divisionen, eine Vereinfachung des Signalverarbeitungsaufwands bedeutet [3] [114]. Die arithmetischen Basisoperationen Addition und Subtraktion sind jedoch in diesem Zahlenformat nicht trivial berechenbar.

3.2 Elementare Funktionen

Elementare Funktionen sind grundlegende mathematische Funktionen, die in einer Vielzahl unterschiedlicher Anwendungsgebiete eingesetzt werden. Ausgehend von der u.a. in [98] gegebenen Definition sind elementare Funktionen eindimensionale Funktionen, die aus einer endlichen Menge an Basisoperationen sowie Logarithmus- und/oder Potenzfunktionen bestehen. Hieraus lassen sich u.a. die Wurzelfunktion oder die Exponentialfunktion ableiten. Da diese Definition komplexe Zahlen beinhaltet, zählen auch trigonometrische Funktionen, z.B. Sinus und Kosinus, hierzu.

Im Bereich des anwendungsspezifischen Schaltungsentwurfs ist die Abbildung elementarer Funktionen essenziell für die Realisierung mathematischer Funktionen und Algorithmen [17] [26]. Ziel ist es dabei, eine möglichst effiziente Hardwareumsetzung zu finden, wobei je nach Anwendungsgebiet unterschiedliche Entwurfskriterien im Vordergrund stehen.

Jede HDL-Implementierung einer elementaren Funktion kann, wie in Kap. 2.4.1 beschrieben, einem Punkt im Hardwareentwurfsraum zugeordnet werden, in dem sich sämtliche möglichen Realisierungen, charakterisiert durch ihre Entwurfskriterien, befinden. Die bestmöglichen Ergebnisse bilden die Pareto-Front. Für die Implementierung elementarer Funktionen ist ein hoher zugrunde liegender Hardwareaufwand für die analytische Berechnung festzustellen. So lassen sich für viele dieser Funktionen aufwändige Rechenverfahren erkennen, z.B. *Restoring*-Algorithmen zur Berechnung der Division [84]. Zudem können Zwischenergebnisse verhältnismäßig große oder kleine Werte annehmen, was mit einer entsprechenden Anpassung der Datenwortbreite oder der Zahlendarstellung einhergeht (siehe Kap. 3.1). Beide Aspekte haben zur Folge, dass eine analytische, hardwarebasierte Abarbeitung elementarer Funktionen zumeist mit geringer Performance einhergeht.

Zur Minimierung dieser Problematik lassen sich zwei Ansätze erkennen: Zum Einen können kritische Terme des Algorithmus durch Umformungen von Gleichungen auf algorithmischer Ebene, z.B. Ausklammern, reduziert werden [129]. Anhand der Kenntnis über die Performance einzelner Operationen wird also eine optimale Hardwareumsetzung ermittelt. Zum Anderen kann umgekehrt eine effiziente Berechnung elementarer Funktionen durch Anpassung der Hardware an die Signalverarbeitung erzielt werden [82]. Kern dieses Ansatzes ist die Berechnung aufwändiger mathematischer Gleichungen, z.B. nicht trivialer, elementarer Funktionen, durch näherungsweise Verfahren, z.B. iterative Algorithmen. Ein vielbeachteter Ansatz zur Steigerung der Performance stellt fehlerbehaftetes Rechnen dar, bei dem ein Rechenergebnis nur näherungsweise genau bestimmt wird [37] [14]. So lassen sich verschiedene Anwendungsgebiete erkennen, bei denen entsprechende Abweichungen, z.B. aufgrund verrauschter Eingangssignale, toleriert werden können. Für diesen Ansatz ist das Entwurfskriterium Rechengenauigkeit (engl.: *accuracy*) von übergeordneter Bedeutung (siehe Kap. 2.4.1). Es müssen dabei

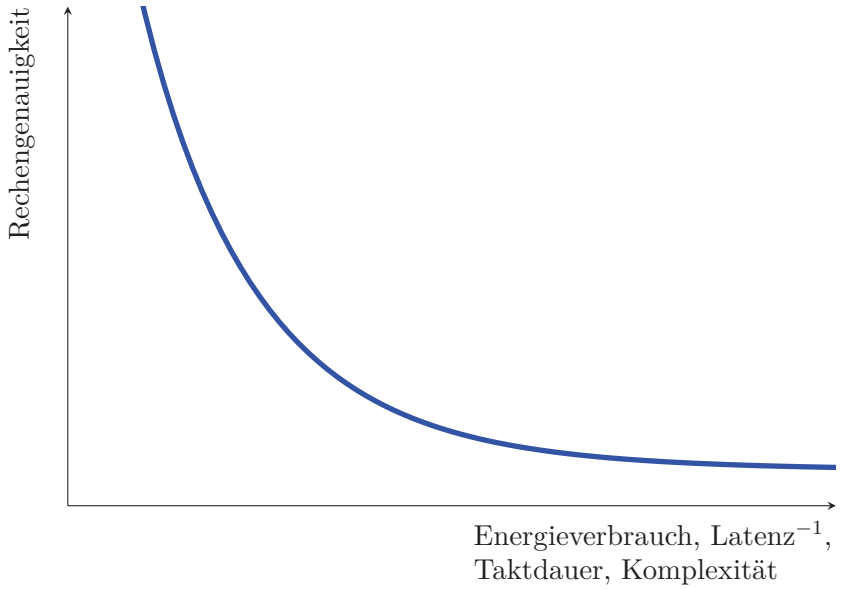


Abbildung 3.3: Hardwareentwurfsraum für näherungsweise Berechnungen mit schematischem Verlauf der Pareto-Front (blaue Linie) für Rechengenauigkeit im Vergleich zu anderen Entwurfskriterien.

je nach betrachteter Anwendung Toleranzbereiche spezifiziert werden, innerhalb derer vernachlässigbare Abweichungen auftreten. Die Auswirkung der Rechengenauigkeit im Hardwareentwurfsraum gegenüber anderen Entwurfskriterien ist schematisch in Abb. 3.3 dargestellt.

Heutzutage existieren viele unterschiedliche Lösungsansätze zur Berechnung elementarer Funktionen [82]. Ein Überblick über verschiedene mathematische Ansätze zur Bestimmung komplexer Funktionen ist in Kap. 3.4, 3.5 und 3.6 gegeben. Nachfolgend wird zunächst die Hardwareumsetzung arithmetischer Basisoperatoren behandelt.

3.3 Arithmetische Basisoperatoren

Die arithmetischen Basisoperatoren stellen die am häufigsten verwendete Rechenvorschrift in digitalen Schaltungen dar, weswegen hierfür eine große Anzahl verschiedener Architekturen existiert [84]. An dieser Stelle sollen ausschließlich gängige, technologieunabhängige Architekturen für stellenwertbasierte Festkommazahlen in Zweierkomplement-Darstellung betrachtet werden.

Addition Die Addition ist der elementare Basisoperator der digitalen Signalverarbeitung. Sämtliche weiteren Operatoren und Funktionen lassen sich aus der Kombination mehrerer Addierer zusammensetzen. Der Aufbau des Addierers bestimmt somit maßgeblich die Performance der resultierenden Schaltung.

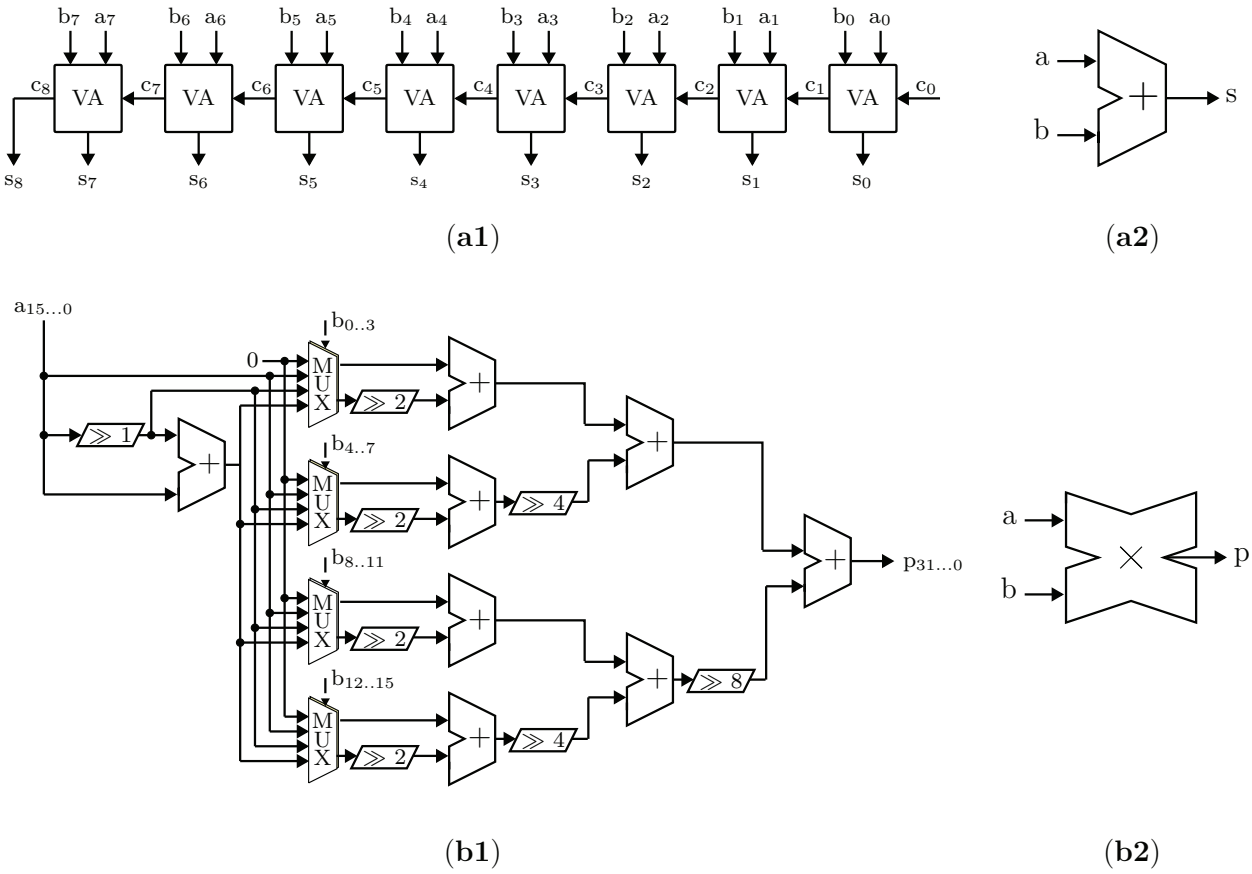


Abbildung 3.4: Hardwareumsetzungen eines **(a1)** 8 Bit CRA bzw. **(b1)** 16 Bit Radix-4-Multiplizierers sowie die allgemeinen Symbole für einen **(a2)** Addierer und **(b2)** Multiplizierer.

Die Standardrealisierung eines digitalen Addieres besteht aus sequenziell verdrahteten Volladdierern. Hierbei werden jedem dieser Module eine Operandenstelle als Eingang zugeordnet und die Übertragsein- und -ausgänge in aufsteigender Reihenfolge zusammengeschaltet. Der Aufbau dieses sogenannten *Carry-Ripple*-Addieres (CRA), der als direkte Umsetzung des schriftlichen binären Additionsverfahrens interpretiert werden kann, ist in Abb. 3.4a1 skizziert. Ein Nachteil dieser Struktur ist die verhältnismäßig hohe Verarbeitungszeit aufgrund der sequenziellen Verschaltung, da der kritische Pfad vom niederwertigsten Eingang bis zum höchstwertigen Ausgang verläuft.

Um diese Standardstruktur zu optimieren, ist es notwendig die Übertragsberechnung zu beschleunigen. Hierzu werden zwei Hilfssignale, das *Generate* g_i und *Propagate* p_i, eingeführt. Nach [90] lassen sich die Überträge durch

$$c_{i+1} = g_i + \sum_{j=0}^{i-1} \left(\prod_{k=j+1}^{i} p_k \right) g_j + \prod_{k=0}^{i} p_k c_0 \tag{3.17}$$

mit

$$g_i = a_i b_i \quad ; p_i = a_i \oplus b_i \tag{3.18}$$

berechnen. Generell werden Addierer, denen eine solche Rechenstruktur zugrunde liegt, als *Carry-Lookahead*-Addierer (CLA) bezeichnet [44]. Es existieren mehrere unterschiedliche Ansätze, z.B. nach Brent-Kung [9] oder nach Kogge-Stone [64], die hinsichtlich ihrer Verschaltung und daraus resultierend ihrer Performance variieren.

Subtraktion Subtraktionen können in der Zweierkomplement-Zahlendarstellung durch einfache Modifikation von Addierern realisiert werden. Allgemein lässt sich jede Subtraktion als Addition eines negierten Zahlenwerts anstelle des Subtrahenden durch

$$y = x_1 - x_2 = x_1 + (-x_2) \tag{3.19}$$

ausdrücken. Die Negierung des zweiten Summanden ergibt sich durch bitweise Invertierung von x_2. Im Zweierkomplement muss zudem der Übertragseingang der ersten Wertstelle auf Eins gesetzt werden ($c_0 = 1$).

Multiplikation Die Multiplikation kann als Abfolge an Additionen angesehen werden, bei der zunächst einzelne Partialprodukte gebildet werden, die in einem zweiten Schritt stellenrichtig aufaddiert werden [84]. Genau wie bei der Umsetzung der Addition gibt es auch für die Multiplikation eine Vielzahl an unterschiedlichen Ansätzen für die Hardwareumsetzung. So ist es möglich, die benötigten Partialprodukte parallel zu berechnen und somit höhere Betriebsgeschwindigkeiten zu erzielen. Darüber hinaus muss bei der Multiplikation die Berechnung unter Berücksichtigung des Vorzeichens geschehen, d.h. für vorzeichenrichtiges Rechnen ist zusätzlicher Hardwareaufwand vonnöten. Da diese Bedingung durch entsprechende Vor- und Nachverarbeitung der Operanden möglich ist, wird im Folgenden zunächst nur die Multiplikation positiver Zahlen beschrieben.

Eine einfache Hardwareumsetzung eines Multiplizierers ist der *Carry-Ripple-Array*-Multiplizierer (CRAM). Hierbei wird beginnend von der niederwertigsten Bitstelle ein Teilprodukt berechnet und zu dem nächst höherwertigen aufaddiert. Das Teilprodukt kann dabei durch eine einfache UND-Verknüpfung bestimmt werden, da das Ergebnis nur dann Eins ist, wenn beide Operandenbits Eins sind.

Neben der linearen Berechnung und Überlagerung der Teilprodukte gibt es zudem mehrere Ansätze, die höhere Parallelität aufweisen. So werden in einem ersten Schritt zunächst alle Teilprodukte vorbestimmt. Diese können dann durch einen als Baumstruktur angeordneten Addierer zusammengefasst werden, womit sich die Verarbeitungsgeschwindigkeit erhöht. Ein typisches Beispiel für solch eine Multipliziererstruktur stellen Radixmultiplizierer dar. Ihre Besonderheit ist die Interpretation der vorlie-

genden Binärzahlen in einem Zahlensystem i.d.R. höherer Ordnung (siehe Kap. 3.1). Beispielhaft soll dieses Prinzip an dieser Stelle für eine Radix-4-Multiplikation erläutert werden: Zunächst werden die Teilprodukte eines Operanden für ein Zahlensystem mit vierwertiger Ziffernmenge ($r = 3$) berechnet, aus denen mit Hilfe eines Multiplexers der aktuell gültige Wert ausgewählt wird. Im Falle eines Radix-4-Multiplizierers müssen also vier mögliche Teilprodukte vorbestimmt werden, was unter Zuhilfenahme von Schiebe- und Addiereroperatoren möglich ist. Die Wortbreite des Operanden, der für die Selektion der Teilprodukte zuständig ist, bestimmt hierbei die Anzahl an Multiplexern. Abschließend müssen die Teilprodukte stellenwertrichtig aufaddiert werden. Eine mögliche Implementierung ist in Abb. 3.4b dargestellt.

Division Allgemein lässt sich die Division als Bestimmung eines Quotienten zweier Zahlen ansehen, was durch den Werteabgleich des Divisors mit Untermengen des Dividenden erreicht wird [90]. Im Vergleich zu den drei anderen Basisoperatoren ist die Hardwareumsetzung der Division deutlich komplexer und unflexibler, weswegen meist versucht wird, die Anzahl an Divisionen in Signalverarbeitungsalgorithmen zu minimieren. Ein Grund hierfür sind die schlechten Parallelisierungseigenschaften der Division, da jeder einzelne Zwischenschritt vom vorangehenden Ergebnis abhängig ist.

Eine einfache Realisierung der Division in Hardware ist der *Restoring*-Dividierer. Hierbei wird ausgehend vom höchstwertigen Bit der Divisor mit der entsprechenden Bitsequenz verglichen, was durch eine Subtraktion umgesetzt werden kann. Ist das Ergebnis positiv, also die Untermenge des Dividenden größer als der Divisor, wird das Ergebnis der Subtraktion übernommen und eine Eins als Ergebnis notiert, da der Dividend im Ergebnis enthalten ist. Andernfalls wird die Untermenge des Dividenden übernommen und eine Null ins Ergebnis übernommen. Nach jedem dieser Teilschritte wird das Zwischenergebnis um das nächstfolgende Bit ergänzt, bis der Dividend vollständig abgearbeitet ist.

3.4 Iterative Näherungsverfahren

Iterative Näherungsverfahren zur Bestimmung elementarer Funktionen basieren auf einer schrittweisen Berechnung des Ergebnisses ausgehend von einem Anfangswert [90]. Der Algorithmus wird dabei in mehrere, sequenziell zu verarbeitende Teilschritte gleicher Rechenvorschrift aufgeteilt. Die Abarbeitungsvorschrift des nachfolgenden Iterationsschritts ist i.d.R. vom vorangehenden Zwischenergebnis abhängig, weswegen sich iterative Näherungsverfahren zumeist nur schlecht parallelisieren lassen. Durch Anwendung der Implementierungstechnik *Loop-Unrolling* bzw. iterative Zerlegung kann allerdings die Latenz bzw. die Komplexität des zugehörigen Datenpfades variiert werden (siehe Kap. 2.3).

3.4.1 Newton-Raphson-Verfahren

Das Newton-Raphson-Verfahren beruht auf dem Prinzip der schrittweisen Bestimmung von Nullstellen einer gegebenen Funktion [110]. Obgleich die Anwendung für eine Vielzahl unterschiedlicher elementarer Funktionen denkbar ist, wird es zumeist zur Berechnung des Kehrwerts und der Wurzelfunktion eingesetzt [48].

Voraussetzung zur Anwendung des Verfahrens ist eine stetig differenzierbare Funktion. Ausgehend von einem Anfangswert wird durch Tangentenbildung ein Nulldurchgang der Funktion abgeschätzt. In besonderen Fällen ist es möglich, dass sich keine Verbesserung der Abschätzung einstellt, d.h. der Algorithmus nicht konvergiert.

Nach [65] kann das Newton-Raphson-Verfahren allgemein durch

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \ , \tag{3.20}$$

mit x_n, x_{n+1} als aktueller und folgender Wert und $f(x_n), f'(x_n)$ als der Funktion und ihrer Ableitung an der Stelle x_n, ausgedrückt werden.

Durch entsprechende Wahl der Funktion $f(x)$ ist es möglich, gegen Funktionswerte elementarer Funktionen zu konvergieren. So ist z.B. für

$$f_1(x) = 1 - \frac{a}{x^2} \tag{3.21}$$

die Nullstelle bei $x = \sqrt{a}$. Eingesetzt in Gl. (3.20) ergibt sich

$$x_{n+1} = \frac{x_n}{2}\left(3 - \frac{x_n^2}{a}\right) \quad ; \quad a > 0 \ , \tag{3.22}$$

womit die Quadratwurzel von a berechnet werden kann. Analog liefert

$$f_2(x) = \frac{1}{x} - a \tag{3.23}$$

eingesetzt in Gl. (3.20) die Rechenvorschrift

$$x_{n+1} = x_n\,(2 - a x_n) \ , \tag{3.24}$$

für die Nullstelle bei $x = \frac{1}{a}$, was die Berechnung des Kehrwerts ermöglicht.

Ein wichtiger Aspekt des Newton-Raphson-Verfahrens ist die Bestimmung des Startwerts x_0, der prinzipiell auch zufällig gewählt werden kann. Dieses Vorgehen kann allerdings in einer hohen Anzahl an Iterationsschritten resultieren oder dazu führen, dass der Algorithmus nicht konvergiert. Daher lässt sich mit einer initialen Wertabschätzung, z.B. durch das Bisektionsverfahren [62], x_0 abschätzen, was jedoch mit zusätzlichem

Rechenaufwand verbunden ist. Eine andere Möglichkeit ist es, einzelne Startwerte für unterschiedliche Funktionsbereiche vorzudefinieren.

Das Newton-Raphson-Verfahren stellt, bezogen auf die algorithmische Umsetzung, eine sehr einfache Methode zur Bestimmung elementarer Funktionen dar. Der Implementierungsaufwand hingegen ist verhältnismäßig hoch, da Multiplizierer und ggf. auch Dividierer benötigt werden (siehe Gl. (3.22)).

3.4.2 CORDIC

Der CORDIC-Algorithmus (engl.: *coordinate rotation digital computer*, CORDIC) erlaubt die Berechnung elementarer Funktionen fast ausschließlich unter Zuhilfenahme von Schiebeoperatoren und Addierern (engl.: *shift-and-add*) [82]. Das Einsatzgebiet war dabei ursprünglich nur auf die Drehung eines Vektors $(x_0, y_0)^T$ um den Winkel φ in Polarkoordinaten begrenzt, wodurch u.a. trigonometrische Funktionen effizient berechnet werden können [120]. Die 1971 vorgestellte verallgemeinerte Form des CORDIC erlaubt zudem die Berechnung von Logarithmus-, Exponential- und Quadratwurzelfunktionen [122]. Insgesamt lassen sich drei verschiedene Rechenmethoden des CORDIC erkennen, die im Folgenden nacheinander, beginnend mit der Drehung in Polarkoordinaten, näher erläutert werden.

Wie vorangehend angesprochen erlaubt der CORDIC die Berechnung trigonometrischer Funktionen durch Vektorrotation. Allgemein lässt sich eine Drehung des Vektors mathematisch mit Hilfe der Drehmatrix $\mathbf{D}_P(\varphi)$ [10] ausdrücken durch

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = \mathbf{D}_P(\varphi) \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \quad \text{mit} \quad \mathbf{D}_P(\varphi) = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{pmatrix} , \tag{3.25}$$

mit x_0, y_0 bzw. x_n, y_n als Koordinaten des originalen bzw. gedrehten Vektors. Unter Verwendung der trigonometrischen Identitäten

$$\cos(\varphi) = \frac{1}{\sqrt{1 + \tan^2(\varphi)}} \quad \text{und} \quad \sin(\varphi) = \frac{\tan(\varphi)}{1 + \tan^2(\varphi)} \tag{3.26}$$

lässt sich die Drehmatrix wie folgt ausdrücken:

$$\mathbf{D}_P(\varphi) = \frac{1}{\sqrt{1 + \tan^2(\varphi)}} \begin{pmatrix} 1 & -\tan(\varphi) \\ \tan(\varphi) & 1 \end{pmatrix} . \tag{3.27}$$

Die Grundidee des CORDIC-Algorithmus ist eine Unterteilung dieser Vektorrotationsgleichung in mehrere Rechenschritte. So wird der Winkel φ allgemein in eine Partial-

summe von n Teilwinkeln unterteilt, was sich unter Berücksichtigung des Vorzeichens durch den Rotationsrichtungskoeffizienten σ_i

$$\varphi \approx z_n = \sum_{i=0}^{n-1} \sigma_i \theta_i \quad \text{mit} \quad \sigma_i \in \{-1, 1\} \ , \tag{3.28}$$

mit z_n als Summe an fest vorgegebenen Teilwinkeln θ_i, ausdrücken lässt. In Polarkoordinaten gilt hierfür

$$\theta_{P,i} = \operatorname{atan}(2^{-i}) \quad ; \quad i = 0, 1, 2, \ldots n-1 \ . \tag{3.29}$$

Der Betrag der Teilwinkel ist somit streng monoton fallend über i. Die in Gl. (3.28) gegebene Aufsummierung konvergiert zudem gegen φ, wobei φ als Grenzwert der Folge angesehen werden kann. Wird dieser bei einem Iterationsschritt überschritten, wird σ_i invertiert, und φ wird von der anderen Seite aus angenähert. Aufgrund der festen Vorgabe der Teilwinkel besitzt das CORDIC-Verfahren standardmäßig einen festen Konvergenzbereich nach [122] von

$$\max |z_{P,n}| \leq \lim_{n \to \infty} \sum_{i=0}^{n-1} \theta_{P,i} = 1,743 \ . \tag{3.30}$$

Verfahren zur Erweiterung des Konvergenzbereichs, wie z.B. in [46] vorstellt, werden aufgrund fehlender Relevanz an dieser Stelle nicht betrachtet.

Unter Berücksichtigung dieser Iterationsvorschrift ergibt sich für die Vektordrehung aus Gl. (3.27)

$$\mathbf{D}_{P,i} = K_{P,i} \begin{pmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{pmatrix} \quad \text{mit} \quad K_{P,i} = \frac{1}{\sqrt{1 + 2^{-2i}}} \ . \tag{3.31}$$

Der CORDIC-Algorithmus kann somit zunächst ausschließlich mit Addierern und Schiebeoperatoren realisiert werden. Durch Ausklammern lässt sich der Skalierungsfaktor K zudem als abschließender Verarbeitungsschritt durch

$$K_{P,n} = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1 + 2^{-2i}}} \tag{3.32}$$

berücksichtigen. Die Berechnung ist also ausschließlich von der Anzahl der Iterationen abhängig und ist für eine gleichbleibende Anzahl an Iterationsschritten konstant. Es ergibt sich

$$\lim_{n \to \infty} K_{P,n} \approx 0,607 \ , \tag{3.33}$$

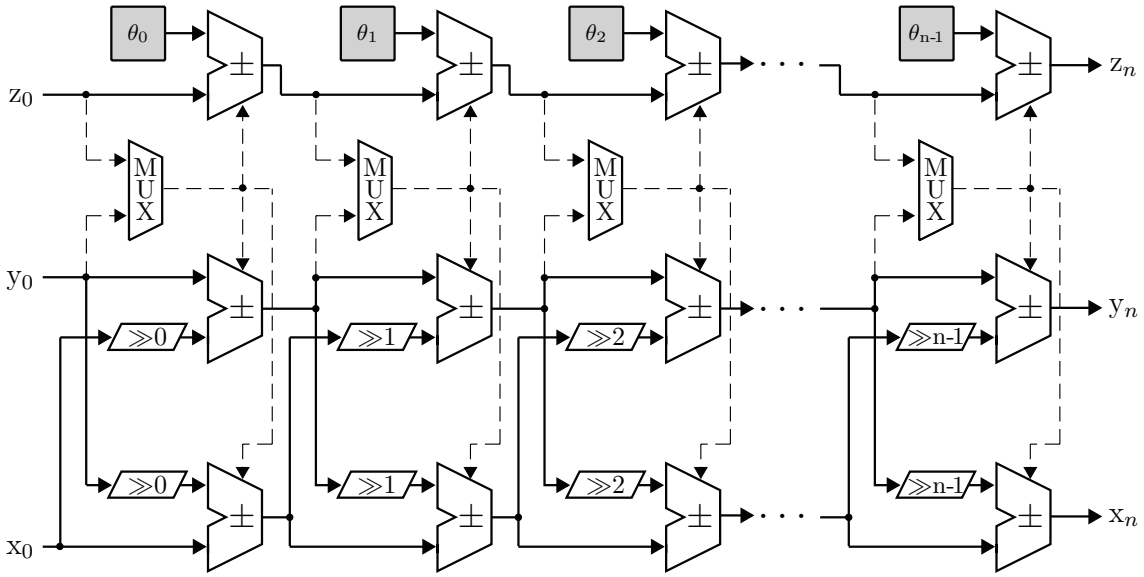


Abbildung 3.5: Prinzipieller Aufbau der CORDIC-Hardwarearchitektur mit n Iterationsschritten zur Vektordrehung in Polarkoordinaten ($m = 1$) nach [77] ohne abschließende Skalierung. Durchgezogene bzw. gestrichelte Linien kennzeichnen den Daten- bzw. Steuerpfad. Ist die explizite Berechnung des Winkels z nicht erforderlich, kann die Struktur auf zwei Addierer pro Iterationsschritt vereinfacht werden, da für die Berechnung von x_n bzw. y_n nur die Drehrichtung bekannt sein muss.

weshalb für CORDIC-basierte Berechnungen allgemein eine abschließende Multiplikation mit $K_{P,n}$ als Korrekturschritt notwendig ist. Für die einfache Bestimmung der Sinus- bzw. Kosinusfunktion vereinfacht sich mit $x_0 = K_{P,n}$ und $y_0 = 0$ Gl. (3.25) zu

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = \begin{pmatrix} \cos(\varphi) \\ \sin(\varphi) \end{pmatrix}, \tag{3.34}$$

so dass die Multiplikation entfällt.

Generell ist es möglich, zwei unterschiedliche Arten der Berechnung mit dem CORDIC-Algorithmus durchzuführen. Neben der bereits eingeführten iterativen, approximativen Vektorrotation, dem Rotationsmodus, kann zudem ein Winkel ausgehend von einem fest vorgegebenen resultierenden Vektor berechnet werden. In diesem sogenannten Vektormodus iteriert die y-Koordinate gegen Null. Der zugehörige Winkel kann über die Aufsummierung der Teilwinkelelemente aus Gl. (3.28) berechnet werden, wobei für jedes φ_i ein Wert z.B. in einem *Lookup-Table* (LUT) hinterlegt werden muss. Für jeden Teilschritt gilt somit

$$z_{P,i+1} = z_{P,i} - \sigma_i \mathrm{atan}(2^{-i}) \ . \tag{3.35}$$

Eine beispielhafte Hardwarearchitektur des CORDIC-Algorithmus findet sich in Abb. 3.5.

Neben der Drehung in Polarkoordinaten besitzt der verallgemeinerte CORDIC zudem hyperbolische und lineare Rechenmethoden. Bei Ersteren werden anstelle von Polarkoordinaten hyperbolische Koordinaten berücksichtigt. Somit verändert sich die Drehmatrix zu

$$\mathbf{D}_H(\varphi) = \begin{pmatrix} \cosh(\varphi) & \sinh(\varphi) \\ \sinh(\varphi) & \cosh(\varphi) \end{pmatrix} , \tag{3.36}$$

was die Berechnung hyperbolischer Funktionen erlaubt. Die damit einhergehende Berechnung einzelner Teilwinkelelemente ergibt sich zu

$$\theta_{H,i} = \operatorname{artanh}(2^{-\delta(i)}) \;\; ; \;\; i = 1, 2, 3, \ldots n-1 \; . \tag{3.37}$$

Die Einführung der Funktion $\delta(i)$ ist notwendig, da in hyperbolischen Koordinaten vereinzelt Iterationsschritte doppelt ausgeführt werden müssen damit der Algorithmus konvergiert (z.B. für $i = 4, 13, 40, \ldots$) [82]. Des Weiteren muss der Skalierungsfaktor an die hyperbolische Drehmatrix angepasst werden. Es gilt

$$K_{H,n} = \prod_{i=1}^{n-1} \frac{1}{\sqrt{1 - 2^{-2\delta(i)}}} \;\; ; \;\; \text{mit} \;\; \lim_{n\to\infty} K_{H,n} \approx 1,207 \; . \tag{3.38}$$

Durch diese Modifikationen ändert sich auch der Konvergenzbereich. So kann in hyperbolischen Koordinaten eine maximale Winkeldrehung von

$$\max |z_{H,n}| \leq \lim_{n\to\infty} \sum_{i=1}^{n-1} \theta_{H,i} = 1,118 \tag{3.39}$$

erreicht werden.

Bei der linearen Rechenmethode werden Multiplikation und Division durch die gegebene CORDIC-Struktur realisiert. Das vorangehend verwendete Prinzip der Koordinatendrehung liegt dieser Methode nicht zugrunde. Vielmehr kann die CORDIC-basierte lineare Funktionsberechnung als iterative Anpassung der unter Kap. 3.3 vorgestellten Ansätze zur Multiplikation und Division auf die durch den CORDIC vorgegebene Struktur verstanden werden. So wird das Ergebnis durch Aufsummieren von Teilprodukten bzw. Teilquotienten iterativ berechnet. Die Teilwinkel werden durch lineare Komponenten

$$z_{L,i+1} = z_{L,i} - \sigma_i 2^{-i} \;\; ; \;\; i = 0, 1, 2, \ldots, n-1 \tag{3.40}$$

ersetzt. Durch diese Abänderung entfällt zum Einen die Berücksichtigung eines Skalierungsfaktors. Zum Anderen umfasst der Konvergenzbereich

$$\max |z_{L,n}| \leq \lim_{n\to\infty} \sum_{i=0}^{n-1} \theta_i = 2 \ . \tag{3.41}$$

Die verallgemeinerten CORDIC-Gleichungen lauten nach [122]

$$\begin{aligned} x_{i+1} &= x_i - m\sigma_i y_i 2^{-\delta(i)} \\ y_{i+1} &= y_i + \sigma_i x_i 2^{-\delta(i)} \qquad \text{, mit } \ m \in \{-1, 0, 1\} \\ z_{i+1} &= z_i - \sigma_i e_i \end{aligned} \tag{3.42}$$

und

$$e_i = \begin{cases} 2^{-i} & \text{, wenn } m = 0 \\ \operatorname{artanh}(2^{-\delta(i)}) & \text{, wenn } m = -1 \\ \operatorname{atan}(2^{-\delta(i)}) & \text{, wenn } m = 1 \end{cases} , \tag{3.43}$$

wiederum mit $m = -1$ als hyperbolische, $m = 0$ als lineare und $m = 1$ als zirkuläre Rechenmethode. Die Berücksichtigung der für $m = 1$ notwendigen doppelten Iterationsschritte erfolgt durch

$$\delta(i) = \begin{cases} i & \text{, wenn } m \geq 0 \\ i - k & \text{, sonst} \end{cases} , \tag{3.44}$$

mit k als größtmögliche ganze Zahl, welche der Bedingung $3^{k+1} + 2^{k-1} \leq 2n$ genügt [82]. Eine Übersicht über die CODRIC-Rechenmethoden ist zudem in Tab. 3.2a dargestellt.

Neben diesen elementaren Funktionen können durch Umformung, spezielle Eingangsdaten oder trigonometrische Identitäten weitere elementare Funktionen realisiert werden. Ein Überblick hierfür findet sich in Tab. 3.2b.

3.5 Polynomiale Approximation

Polynomiale Approximation beruht auf der Nachbildung elementarer Funktionen durch Polynome [5]. Im Gegensatz zu iterativen Verfahren wird die Genauigkeit der Näherung nicht durch die Anzahl der Iterationsschritte bestimmt, sondern durch den Grad n des

Tabelle 3.2: Elementare Funktionen des verallgemeinerten CORDIC-Algorithmus nach [90]. Die Basisfunktionalität der verschiedenen Rechenmethoden ist in **(a)** dargestellt. Durch spezielle Wahl der Eingangswerte lassen sich die elementaren Funktionen in **(b)** berechnen.

Rechenmethode	m	Vektor	Rotation
zirkulär	1	$x_n = \sqrt{x_0^2 + y_0^2}$ $z_n = z_0 + \operatorname{atan}(\frac{y_0}{x_0})$	$x_n = x_0 \cos(z_0) - y_0 \sin(z_0)$ $y_n = y_0 \cos(z_0) + x_0 \sin(z_0)$
linear	0	$x_n = x_0$ $z_n = z_0 + \frac{y_0}{x_0}$	$x_n = x_0$ $y_n = y_0 + x_0 z_0$
hyperbolisch	-1	$x_n = \sqrt{x_1^2 - y_1^2}$ $z_n = z_1 + \operatorname{artanh}(\frac{y_1}{x_1})$	$x_n = x_1 \cosh(z_1) + y_1 \sinh(z_1)$ $y_n = y_1 \cosh(z_1) + x_1 \sinh(z_1)$

(a)

Funktion		Gleichung	Funktion		Gleichung
e^z	=	$\cosh(z_1) + \sinh(z_1)$	e^{-z}	=	$\cosh(z_1) - \sinh(z_1)$
$\tan(\alpha)$	=	$\frac{\sin(\alpha)}{\cos(\alpha)}$	$\sqrt{\alpha}$	=	$\sqrt{\left(\alpha + \frac{1}{4}\right)^2 - \left(\alpha - \frac{1}{4}\right)^2}$
$\ln z$	=	$2\operatorname{artanh}(\frac{\alpha-1}{\alpha+1})$	$\operatorname{artanh}(\alpha)$	=	$\frac{\sinh(\alpha)}{\cosh(\alpha)}$

(b)

Polynoms sowie die verwendeten Koeffizienten a_n. Allgemein ist ein Polynom $P(x)$ definiert durch

$$P(x) = \sum_{k=0}^{n} a_k x^k = a_n x^n + a_{n-1} x^{n-1} \ldots a_1 x^1 + a_0 \ . \tag{3.45}$$

Für die Approximation einer elementaren Funktion $f(x)$ gilt

$$f(x) \approx P(x) = \sum_{k=0}^{n} a_k x^k \ . \tag{3.46}$$

Je höher der Grad des Polynoms, desto besser ist i.d.R. die Näherung für $f(x)$. Für eine mögliche Hardwareumsetzung bedeutet dies einen direkten Zusammenhang zwischen der algorithmischen Komplexität und der Güte der Approximation. So besteht der zugehörige digitale Schaltungsaufbau aus untereinander verdrahteten Addierer- und Multiplizierermodulen. Ein Beispiel dafür ist in Abb. 3.6 gegeben. Für die Polynomapproximation muss, wie bei den iterativen Verfahren aus Kap. 3.4, der Wertebereich

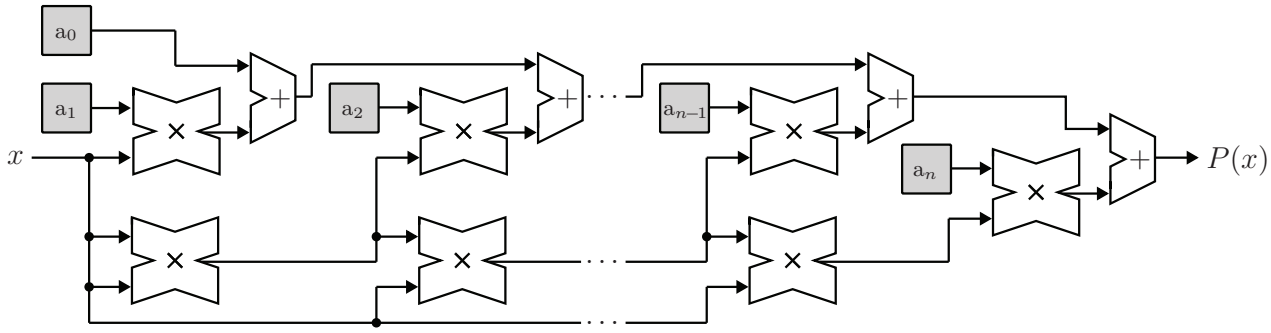


Abbildung 3.6: Hardwareumsetzung einer polynomialen Funktion $P(x)$ n-ten Grades nach Gl. (3.45).

vorab berücksichtigt werden. So ist es bei diesem Verfahren notwendig einen Funktionsabschnitt auszuwählen, der approximiert werden soll. Auch hierbei besteht ein direkter Zusammenhang zwischen der Länge eines solchen Funktionssegmentes und den benötigten Hardwareressourcen. Allgemein wird die Beschaffenheit eines Polynoms durch seine Koeffizienten festgelegt. Um für die Originalfunktion $f(x)$ eine polynombasierte Näherung zu bestimmen, lässt sich eine Vielzahl an unterschiedlichen Ansätzen erkennen, die sich zumeist in der Art der Koeffizientenbestimmung unterscheiden [33]. Einige populäre Ansätze werden nachfolgend vorgestellt.

3.5.1 Polynomiale Regression

Bei der Verwendung von Regressionsverfahren wird eine Funktion anhand vorgegebener Stützstellen $\tilde{\mathbf{x}}, \tilde{\mathbf{y}}$ gebildet. Ziel ist es dabei, einen mathematischen Zusammenhang zwischen den Eingangsvariablen $\mathbf{x}$, den sogenannten Regressoren, zu einer Ausgangsgröße zu formulieren, der einen angenäherten Funktionsverlauf beschreibt. Dabei ist es nicht zwingend notwendig, dass sämtliche Stützstellen exakt abgebildet werden. Die zugehörige Funktionsgleichung kann durch

$$f(\mathbf{x}) = f_R(\mathbf{x}) + \varepsilon \quad \approx \quad f_R(\mathbf{x}) \quad ; \mathbf{x} = \begin{pmatrix} x_0 \\ \vdots \\ x_n \end{pmatrix} , \tag{3.47}$$

mit ε als Restglied und x_i als Eingangsvariable, beschrieben werden. Bei der in dieser Arbeit betrachteten polynomialen Regression elementarer Funktionen vereinfacht sich die Approximation zum eindimensionalen Ausdruck

$$f_{PR}(x) = \sum_{k=0}^{n} a_k x^k. \tag{3.48}$$

Zur Bestimmung der Koeffizienten muss für jede Stützstelle eine zugehörige polynomiale Gleichung formuliert werden, wodurch das lineare Gleichungssystem

$$\underbrace{\begin{pmatrix}\tilde{y}_0\\ \vdots\\ \tilde{y}_n\end{pmatrix}}_{\tilde{\mathbf{y}}} = \underbrace{\begin{pmatrix}1 & \tilde{x}_0^1 & \dots & \tilde{x}_0^k\\ \vdots & & & \\ 1 & \tilde{x}_n^1 & \dots & \tilde{x}_n^k\end{pmatrix}}_{\tilde{\mathbf{X}}} \underbrace{\begin{pmatrix}\hat{a}_0\\ \vdots\\ \hat{a}_n\end{pmatrix}}_{\hat{\mathbf{a}}} + \underbrace{\begin{pmatrix}\varepsilon_0\\ \vdots\\ \varepsilon_n\end{pmatrix}}_{\boldsymbol{\varepsilon}} \tag{3.49}$$

entsteht. Ausgehend hiervon kann mittels unterschiedlicher mathematischer Verfahren ein Lösungsvektor $\mathbf{a}$ bestimmt werden, was auch als Schätzung bezeichnet wird [82]. Im Rahmen dieser Arbeit soll ausschließlich die Methode der kleinsten Quadrate (KQ) betrachtet werden. Ihre grundlegende Idee beruht auf der Minimierung der Summe der quadrierten Fehler. Nach [27] ist die Berechnungsvorschrift für KQ durch

$$KQ(\hat{\mathbf{a}}) = \left(\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\hat{\mathbf{a}}\right)^T \left(\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\hat{\mathbf{a}}\right) \tag{3.50}$$

gegeben. Durch Umformen und Differenzieren folgt

$$\frac{\partial}{\partial \hat{\mathbf{a}}} KQ(\hat{\mathbf{a}}) = -2\tilde{\mathbf{X}}^T\tilde{\mathbf{y}} + 2\tilde{\mathbf{X}}^T\tilde{\mathbf{X}}\hat{\mathbf{a}} \ . \tag{3.51}$$

Die Lösung ergibt sich mit $\frac{\partial}{\partial \hat{\mathbf{a}}} KQ(\hat{\mathbf{a}}) \stackrel{!}{=} 0$ zu

$$\mathbf{a} = \left(\tilde{\mathbf{X}}^T\tilde{\mathbf{X}}\right)^{-1} \tilde{\mathbf{X}}^T\tilde{\mathbf{y}} \ . \tag{3.52}$$

Eine umfassende Herleitung findet sich u.a. in [27].

Um die polynomiale Regression zur Approximation elementarer Funktionen einzusetzen, muss die Originalfunktion zunächst quantisiert werden. Ausgehend von den so entstehenden Stützstellen lassen sich die benötigten Polynomkoeffizienten a_i durch Gl. (3.52) schätzen.

3.5.2 Taylor-Näherung

Taylorpolynome $T(x)$ stellen eine spezielle Lösung der vorangehend eingeführten polynomialen Regression dar, bei der die Koeffizienten über einen nichtlinearen KQ-Ansatz bestimmt werden [30]. Hieraus folgt eine allgemeine Rechenvorschrift, bei der die Näherung durch Ableitung der Originalfunktion $f(x)$ an nur einer Stützstelle $\tilde{x}$ gebildet werden kann. Voraussetzung ist, dass $f(x)$ eine $(n+1)$-mal stetig differenzierbare Funktion ist. Allgemein ist die Taylor-Näherung unter Berücksichtigung von Gl. (3.48) durch

$$f(x) = T_n(x) + \varepsilon \approx T_n(x) \ , \tag{3.53}$$

mit $T_n(x)$ als Taylorpolynom n-ten Grades und ε als zugehöriges Restglied, beschrieben. $T_n(x)$ ist dabei definiert als

$$\begin{aligned} T_n(x) &= \sum_{k=0}^{n} \frac{f^{(k)}(\tilde{x})}{k!}(x-\tilde{x})^k \\ &= f(\tilde{x}) + \frac{f'(\tilde{x})}{1!}(x-\tilde{x}) + \frac{f''(\tilde{x})}{2!}(x-\tilde{x})^2 + \ldots \frac{f^{(n)}(\tilde{x})}{n!}(x-\tilde{x})^n \ , \end{aligned} \tag{3.54}$$

wobei der rechte Term als Taylorreihe bezeichnet wird. Je größer n, desto geringer ist der Anteil des Restgliedes, womit eine verbesserte Funktionsapproximation von $f(x)$ erzielt wird. Die Koeffizienten des Taylorpolynoms lassen sich dabei direkt über die Ableitungen an der vorgegebenen Stelle $\tilde{x}$ bestimmen.

3.5.3 Multipliziererfreie Polynome

Die multipliziererfreie Umsetzung von Polynomen ist ein allgemeiner Ansatz zur Reduktion des Signalverarbeitungsaufwands. Ausgangspunkt dieses Verfahrens sind Polynome ersten Grades, also Geradengleichungen, die durch

$$f_{li}(x) = a_1 x + a_0 \ , \tag{3.55}$$

mit a_1 als Steigung und a_0 als Achsenverschiebung, definiert sind. Um den Hardwareaufwand weiter zu reduzieren, muss der Koeffizient a_1 spezielle Anforderungen erfüllen, durch welche die Multiplikation signifikant vereinfacht wird. Wie in Kap. 3.3 gezeigt, berechnen digitale Multiplizierer i.d.R. ihr Ergebnis durch Aufsummierung von Teilprodukten. Durch Begrenzung der maximalen Anzahl der partiellen Produkte kann der Signalverarbeitungsaufwand deutlich verringert werden. Dieses Vorgehen kann somit als quantisierte Multiplikation angesehen werden. Die Anzahl der verwendeten Teilprodukte kann dabei über den Quantisierungsfaktor (QF) angegeben werden. Um dieses Verfahren so flexibel wie möglich zu gestalten, können partielle Produktterme auch voneinander subtrahiert werden. Unter Berücksichtigung dieser Randbedingung ergibt sich die multipliziererfreie Geradengleichung

$$f_{MF}(x) = \left(\sum_{j=0}^{n-1} \pm 2^{\lambda_j} x \right) + a_0 \ ; \ \lambda_j \in \mathbb{Z} \ , \tag{3.56}$$

mit n als Quantisierungsfaktor und 2^{λ_j} als j-tes partielles Produkt. Der Rechenaufwand der Approximation ist also beschränkt auf Schiebeoperatoren und Additionen. Aufgrund dieser starken Einschränkungen ist ein sinnvoller Einsatz dieser Methode i.d.R. nur für kleine Funktionsbereiche, wie sie z.B. in Kap. 3.6 vorkommen, möglich.

3.6 Tabellengestützte Verfahren

Das grundlegende Prinzip tabellengestützter Verfahren ist die Unterteilung der Originalfunktion $f(x)$ in Unterbereiche [82]. Innerhalb dieser sogenannten Segmente kann der zugehörige Funktionsabschnitt auf unterschiedliche Weise angenähert werden. Die dafür benötigten Informationen, z.B. Zahlenwerte der Konstanten, werden tabellarisch, d.h. in Speicherzellen oder LUTs, hinterlegt. Je höher dabei die Anzahl an Segmenten, desto größer ist auch der Speicherbedarf. Andererseits geht hiermit auch i.d.R. eine deutliche Reduktion der Ausführungsgeschwindigkeit einher, da für kleinere Segmente zumeist einfachere Verfahren der Funktionsapproxmiation eingesetzt werden können. Neben der Segmentierung ist bei tabellengestützten Verfahren auch die Wahl einer geeigneten Approximationsmethodik relevant.

Allgemein lassen sich tabellengestützte Verfahren durch

$$f(x) \approx \tilde{f}_T(x) = \begin{cases} f_{T,1}(x) & \text{seg}(0) \leq x < \text{seg}(1) \\ f_{T,2}(x) & \text{seg}(1) \leq x < \text{seg}(2) \\ \quad \vdots & \\ f_{T,k}(x) & \text{seg}(k-1) \leq x < \text{seg}(k) \end{cases}, \tag{3.57}$$

mit $f_{T,i}(x)$ als i-te Teilfunktion und $\text{seg}(i)$ als Intervallgrenze zwischen dem $(i-1)$-ten und dem i-ten Segment sowie k als Anzahl an Segmenten, ausdrücken. Nach [82] sind verschiedene Ansätze zur Umsetzung tabellengestützter Verfahren erkennbar, von denen zwei im Folgenden vorgestellt werden.

3.6.1 Exakte Verfahren

Exakte Verfahren basieren auf der vollständigen Abbildung aller benötigten Funktionswerte in ein Speichermodul [82]. Diese Art der Funktionsabbildung ergibt sehr präzise Näherungen elementarer Funktionen, da Ungenauigkeiten allein auf Quantisierungseffekte des Zahlenformats zurückzuführen sind. Durch Interpretation des Operanden als Speicheradresse ist es zudem möglich, mit geringem zusätzlichen Aufwand eine elementare Funktion allein durch ein Speichermodul abzubilden.

Obgleich dieser Ansatz damit bestmögliche Resultate hinsichtlich der Rechengenauigkeit erzielt, lassen sich jedoch auch hohe Anforderungen bezüglich Komplexität und Energieverbrauch erkennen. So müssen für einen n Bit breiten Operanden 2^n Werte für $f(x)$ zur Verfügung stehen, also z.B. bei 16 Bit 64 kByte Speicher. Abhängig von $f(x)$ kann allerdings in einigen Fällen der Speicherbedarf deutlich reduziert werden: Weist eine elementare Funktion Symmetrieeigenschaften auf, so kann durch entsprechende Anpassung der Operanden beim Speicherzugriff der Platzbedarf verringert werden.

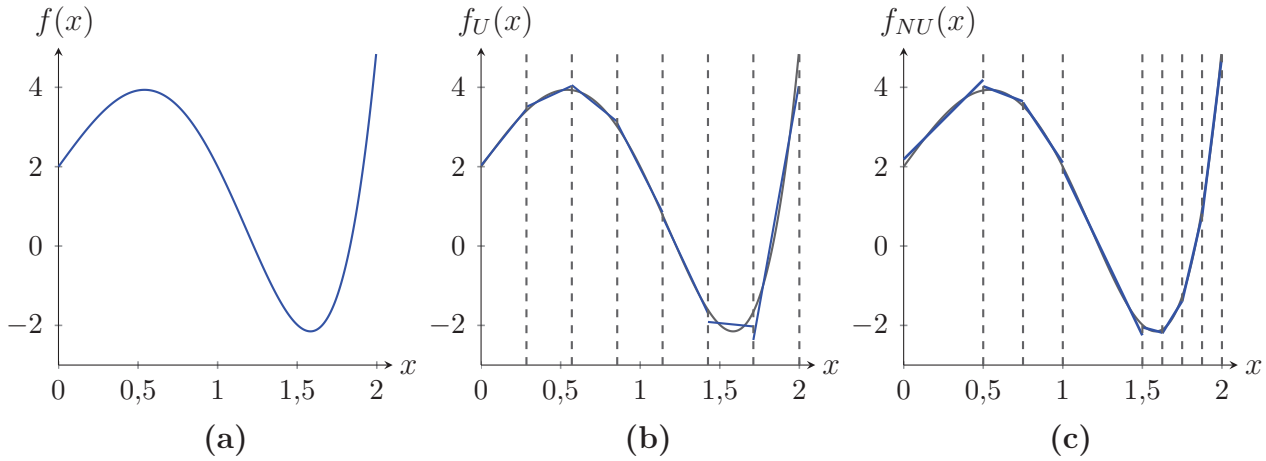


Abbildung 3.7: Approximation der Funktion $f(x) = \frac{3}{2}x^5 - 7x^3 + \frac{11}{2}x + 2$ im Bereich $0 \leq x < 2$. **(a)** zeigt den Verlauf der Originalfunktion, **(b)** bzw. **(c)** eine mögliche uniforme bzw. nicht-uniforme Segmentierung mit linearer Approximation. Die Teilfunktionen in den einzelnen Segmenten wurden durch lineare Regression (siehe Kap. 3.5.1) ermittelt.

3.6.2 Rechengestützte Verfahren

Bei rechengestützten Verfahren werden den durch Unterteilung entstehenden Segmenten Teilfunktionen zugeordnet, die mit Hilfe vorangehend vorgestellter Verfahren aus Kap. 3.4 und 3.5 approximiert werden können. So werden oftmals Polynome niederen Grades verwendet, um gleichbleibende Ergebnisse hinsichtlich Genauigkeit im Vergleich zum allgemeinen Polynomansatz aus Kap. 3.5.1 zu erzielen.

Um den Aufwand zur Ansteuerung des aktiven Segmentes möglichst hardwarefreundlich zu halten, muss die Bestimmung der Intervallgrenzen speziellen Anforderungen genügen. Allgemein werden dabei zwei unterschiedliche Ansätze betrachtet: uniforme und nicht-uniforme Segmentierung [23].

Bei uniformer Segmentierung sind die Intervallgrenzen durch

$$\text{seg}(i) = \text{seg}(0) + i \cdot 2^{\lambda} \quad ; \quad \lambda \in \mathbb{Z} \,, \tag{3.58}$$

mit seg(0) als Anfangswert des Funktionsbereichs und 2^{λ} als Länge eines Segments, festgelegt. Die Gesamtlänge der Originalfunktion muss der Bedingung

$$\text{seg}(k) - \text{seg}(0) = 2^{\lambda_{max}} \quad ; \quad \lambda_{max} \in \mathbb{Z} \,, \tag{3.59}$$

mit k als Anzahl an Segmenten, genügen. Durch diese Vorgabe kann das aktive Segment durch Betrachtung der höchstwertigen Bits (engl.: *most significant bit*, MSB) ausgewählt werden. Die MSBs können somit direkt als Adresswort zur Ansteuerung

des LUT genutzt werden, womit kein zusätzlicher Bedarf an Ansteuerlogik anfällt. Ein Beispiel für uniforme Segmentierung ist in Abb. 3.7b visualisiert.

Eine weitere Möglichkeit zur Unterteilung der Originalfunktion stellt die nicht-uniforme Segmentierung dar. Hierbei können einzelne Segmente unterschiedlich groß sein. Für eine möglichst hardwarefreundliche Umsetzung werden die Intervallgrenzen wie folgt festgelegt

$$\mathrm{seg}(i) = \mathrm{seg}(0) + \sum_{i=0}^{k-1} 2^{\lambda_i} \quad ; \quad \lambda_i \in \mathbb{Z} \ . \tag{3.60}$$

Für jedes Segment i kann die Segmentlänge somit variieren. Da auch bei diesem Ansatz jede Intervallgröße als Zweierpotenz realisiert ist, findet der Segmentzugriff mit geringem Hardwareaufwand statt. So kann die Intervallgrenze, ähnlich der uniformen Segmentierung, direkt aus den MSBs abgeleitet werden. Je nach Segmentgröße variiert dabei auch die Länge der MSBs, was zu speziellen Randbedingungen für die Segmentanordnung führt. Ein automatisiertes Vorgehen hierfür wird in Kap. 3.7 vorgestellt. Abb. 3.7c zeigt ein allgemeines graphisches Beispiel für nicht-uniforme Segmentierung.

3.7 Automatische lineare Funktionsapproximation

Automatische lineare Funktionsapproximation (ALFA) stellt eine im Rahmen der Dissertation entwickelte und verwendete Erweiterung und Kombination der vorangehend vorgestellten Verfahren aus Kap. 3.5.1, Kap. 3.5.3 und Kap. 3.6.2 dar. Hierbei werden Parameter, wie z.B. Funktionskoeffizienten oder Anfangswerte einer Iteration, automatisch in den Prozess der Funktionsapproximation integriert. Obgleich dies prinzipiell für eine Vielzahl unterschiedlicher Approximationsverfahren angewendet werden kann, werden im Rahmen dieser Arbeit ausschließlich tabellengestützte Verfahren mit nicht-uniformen, multipliziererfreien Polynomen verwendet [103]. Ausgehend von einer fest vorgegebenen Rechengenauigkeit müssen zunächst Position und Größe der benötigten Intervalle gemäß diesen Vorgaben bestimmt werden. Hieraus ergibt sich die mathematische Beschreibung

$$\tilde{f}(x) = \left[\sum_{j=0}^{l-1} \pm \begin{pmatrix} 2^{\lambda_{0,j}} \\ 2^{\lambda_{1,j}} \\ \vdots \\ 2^{\lambda_{k-1,j}} \end{pmatrix} x + \boldsymbol{a}_0 \right] \cdot \boldsymbol{\kappa}(x) \ ; \ \lambda_{i,j} \in \mathbb{Z} \ , \tag{3.61}$$

mit $2^{\lambda_{i,j}}$ als quantisierter Steigung, l als Quantisierungsfaktor und $\boldsymbol{a}_0$ als Achsenabschnittsvektor. $\boldsymbol{\kappa}(x)$ ist die Segmentierungsfunktion, die durch

$$\boldsymbol{\kappa}(x) = \begin{cases} (1,0,...,0)^T; & \text{seg}(0) \leq x < \text{seg}(1) \\ (0,1,...,0)^T; & \text{seg}(1) \leq x < \text{seg}(2) \\ & \vdots \\ (0,0,...,1)^T; & \text{seg}(k-1) \leq x < \text{seg}(k) \end{cases} \tag{3.62}$$

das aktuelle Segment selektiert. seg(0), seg(k) bzw. seg(i) sind Start- und Endpunkt von $\tilde{f}(x)$ bzw. vom i-ten Segment. Unter Einbeziehung von Gl. (3.59) und Gl. (3.60) ist es somit möglich, hardwarefreundliche, nicht-uniforme Segmentierung durchzuführen.

Die ALFA-basierte Funktionsapproximation kann als automatisierter Syntheseschritt für den Hardwareentwurf elementarer Funktionen angesehen werden. Ausgehend von einer algorithmischen Beschreibung wird durch Spezifikation der Rechengenauigkeit eine RTL-Schaltungsbeschreibung erzeugt (siehe Abb. 3.8). Je höher dabei die gewünschte Rechengenauigkeit, desto schlechter ist i.d.R. die Performance (siehe Kap. 3.2).

Allgemein setzt sich die ALFA-basierte Funktionsapproximation aus zwei unterschiedlichen Schritten zusammen: der Parameterextraktion, also der eigentlichen algorithmischen Bestimmung von Koeffizienten der zu approximierenden Funktion, und der Übersetzung in eine HDL-Implementierung. Da die behandelte Funktion automatisch approximiert werden soll, ist für beide Schritte eine entsprechende Heuristik bzw. generische Abarbeitung notwendig. Somit wird im Folgenden der generelle Ablauf der Parameterextraktion und der HDL-Erzeugung erläutert, die für die ALFA-Synthese in Form von Matlab- bzw. Java-Programmen entwickelt wurde, und die automatische Bearbeitung realisieren.

3.7.1 Parameterextraktion

Die Parameterextraktion ermöglicht die Approximation einer elementaren Funktion zum Einen durch Vorgabe von allgemeinen Randbedingungen wie Rechengenauigkeit oder Quantisierung der Multiplikation (siehe Kap. 3.5.3). Zum Anderen ist es notwendig, die Beschaffenheit der Operanden, wie digitale Auflösung oder Zahlenformat, näher zu spezifizieren, wobei Letzteres auf das Q-Format beschränkt ist. Ausgehend von diesen Parametern wird eine Funktion $f(x)$ in einem fest vorgegebenen Intervall durch lineare, multiplizierfreie Gleichungen angenähert. Dafür wird zunächst eine entsprechende Geradengleichung als Referenzapproximation von $f(x)$, mit Hilfe linearer Regression, erzeugt (siehe Kap. 3.5.1). Der quantisierte Multiplikator $a_{1,i}$ wird anhand des Parameters QF erstellt, wobei dieser so gewählt wird, dass die Differenz zur Stei-

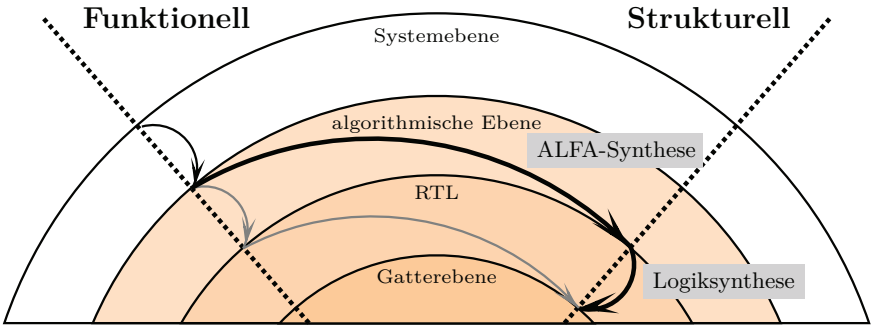


Abbildung 3.8: Automatische lineare Funktionsapproximation (ALFA) als Hardwareentwurfswerkzeug im Gajski-Diagramm. Anstelle des in Kap. 2.4.2 beschriebenen Ansatzes, bei dem ein mathematischer Term zunächst in der funktionalen Sicht in eine RTL-Hardwarebeschreibung übertragen wird, findet bei ALFA direkt eine Transformation, unter Berücksichtigung einer spezifizierten Rechengenauigkeit, in eine strukturelle Hardwarebeschreibung auf RT-Ebene statt. Die grauen bzw. schwarzen Pfeile stellen somit das herkömmliche bzw. ALFA-basierte Verfahren dar.

gung der Regressionsgeraden minimal ist. Hieran schließt sich die Bestimmung des Achsenabschnittskoeffizienten durch die Rechenvorschrift

$$a_{0,i} = \arg\min_{\hat{a}_{0,i}} \left(||\mathbf{f}_i - (a_{1,i} \cdot \mathbf{x}_i + \hat{a}_{0,i})||_1 \right) \ , \tag{3.63}$$

mit $\mathbf{f}_i$ bzw. $\mathbf{x}_i$ als Werte der Teilfunktion bzw. Definitionsbereich des i-ten Segments sowie $\hat{a}_0$ als Laufvariable möglicher Koeffizienten und $||\,\centerdot\,||_1$ als L^1-Norm, an.

Im nächsten Schritt wird die Approximation mit der Originalfunktion verglichen. Unter Berücksichtigung der Quantisierung des Definitionsbereiches wird hierfür der mittlere absolute Fehler (engl.: *mean absolute error*, mae), wie u.a. in [49] gegeben, durch

$$\text{mae} = \frac{1}{n} ||\mathbf{f}_i - \tilde{\mathbf{f}}_i||_1 \ , \tag{3.64}$$

mit n als Anzahl an Eingangsvariablen und $\tilde{\mathbf{f}}_i$ als approximierte Teilfunktion des i-ten Segments, berechnet. Ist die Abweichung der Rechengenauigkeit dabei größer als der vorgegebene, spezifizierte Wert, werden zwei Teilfunktionen mit jeweils halber Anzahl an Eingangsvariablen durch Bisektion erstellt. Durch die in Gl. (3.60) formulierte Bedingung lassen sich die Teilfunktionen mit Hilfe der MSBs selektieren. Danach wird erneut eine Funktion f_{lin} nach dem vorangehend beschriebenen Verfahren für die

Algorithmus 3.1: Automatische lineare Funktionsapproximation

```
void ALFA(double[] f(x), short qf, double[] x, double mae_ALFA) {
// Originalfunktion f(x), QF qf, Wertemenge x, Rechengenauigkeit mae_ALFA
  short k, i = 0;
  double[] f_i = f(x);
  double[] x_i = x;
  do {
// Berechnung der aktuellen linearen Approximation (siehe auch Anhang A.1)
    double a_1,i = estimateQuantizedGradient(f_i, qf);
    double a_0,i = estimateBias(a_1,i, f_i, x_i);
// Berechnung des mittleren absoluten Fehlers
    double mae_i = abs((a_1,i · x_i + a_0,i)-f_i) / (getLength(f_i)-1);
// Auswertung der aktuellen Rechengenauigkeit
    if (mae_i > mae_ALFA) {
      f_i = getSubvector(f_i, 0, getLength(f_i/2)-1);
      x_i = getSubvector(x_i, 0, getLength(x_i/2)-1);
      k++;
    } else {
      f_i = nextSegment(f(x));
      x_i = nextSegment(x);
      i++;
    }
  } while (i < k);
}
```

niederwertige (linke) Teilfunktion berechnet. Je nach Anzahl der Rekursionsschritte, verringert sich dabei die Größe des zu approximierenden Bereichs.

Wird die spezifizierte Rechengenauigkeit für einen Funktionsabschnitt eingehalten, fährt der Algorithmus sukzessive mit dem nächst höherwertigen (rechts folgenden) Segment fort, bis die Originalfunktion vollständig abgearbeitet ist. Da nicht jede Teilfunktion über die gleiche Anzahl an Rekursionsschritten bestimmt werden muss und somit die zugehörigen Funktionsbereiche in der Länge variieren können, entstehen i.d.R. nichtuniforme Segmente. Dennoch ist die Auswahl der Segmente vergleichsweise einfach über die Betrachtung der MSBs möglich. Die Anzahl der MSBs entspricht dabei der Anzahl der jeweiligen Rekursionsschritte. Eine Pseudocodebeschreibung dieser Heuristik zeigt Alg. 3.1 (siehe auch Anhang A.1); ein graphisches Beispiel für die Segmentierung mit unterschiedlichen Rechengenauigkeiten und variierendem QF ist in Abb. 3.9 dargestellt.

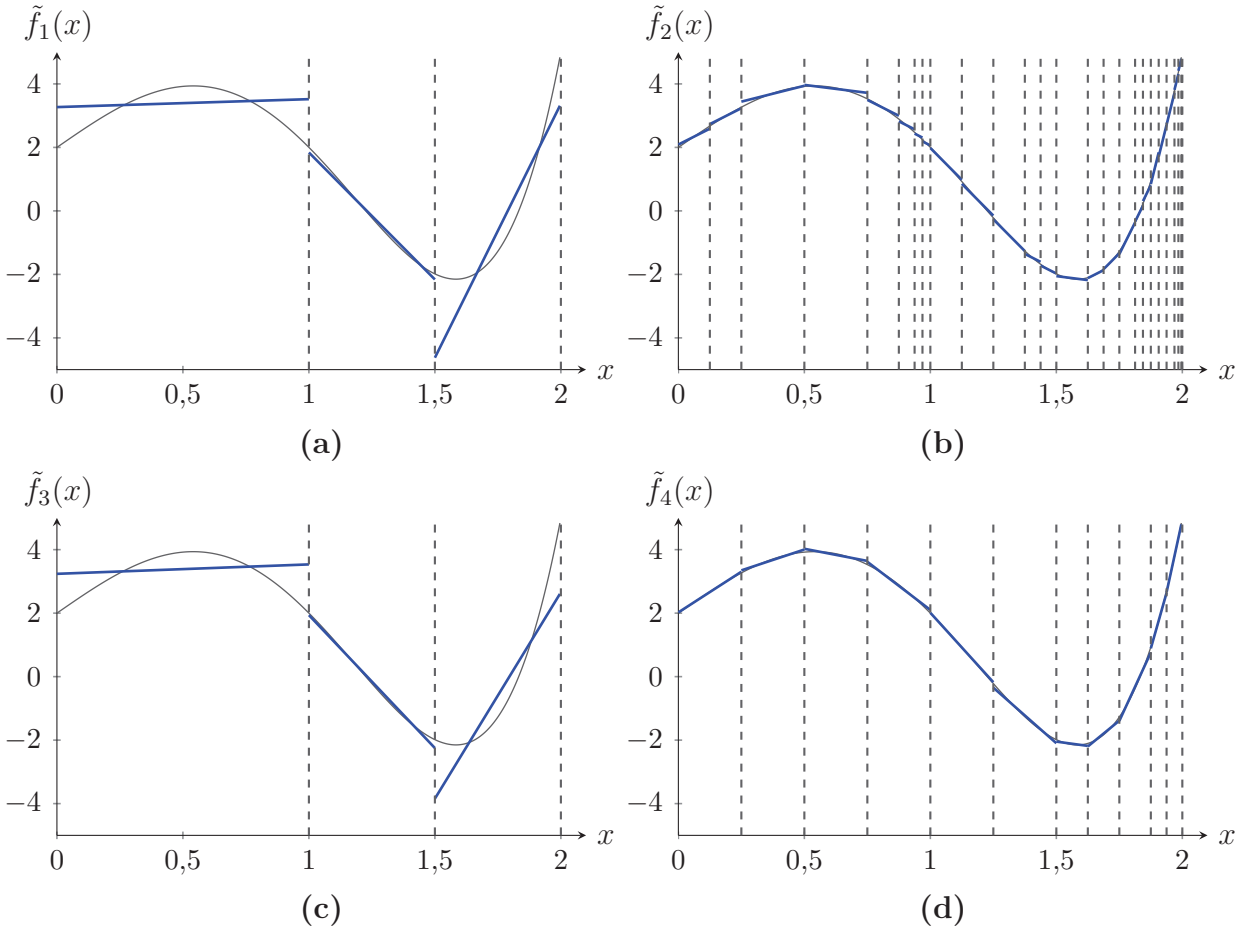


Abbildung 3.9: Beispiel ALFA-basierter Funktionsapproximationen in Abhängigkeit vom Quantisierungsfaktor und mae mit **(a)** $\text{mae}_{ALFA} = 1$, QF$= 1$, **(b)** $\text{mae}_{ALFA} = 0,05$, QF$= 1$, **(c)** $\text{mae}_{ALFA} = 1$, QF$= 3$ und **(d)** $\text{mae}_{ALFA} = 0,05$, QF$= 3$. Für $\text{mae}_{ALFA} = 1$ wird in beiden Fällen die gleiche Anzahl an Segmenten benötigt. Im zweiten Fall ist für QF$= 1$ ein höherer Segmentierungsaufwand zu erkennen.

Eine Besonderheit der vorgestellten Parameterextraktion ist das Auftreten von Sprungstellen an den Intervallgrenzen der einzelnen Segmente. Obgleich hierdurch eine möglichst genaue Approximation der aktuellen Teilfunktion gewährleistet ist, kann diese Eigenschaft in einigen Anwendungen, z.B. Regelkreisen, zu fehlerhaftem Verhalten, z.B. Aufschwingen, führen [96]. Um dieses Problem zu umgehen kann die Parameterextraktion zusätzlich stetige Approximationen generieren. So ist der Aufpunkt des letzten Wertepaares $P(x_{P,i}, y_{P,i})$ der vorangehenden Funktion hierbei der Bezugspunkt der nachfolgenden Approximationsgeraden. Über den allgemeinen Zusammenhang

$$\tilde{f}(x) = a_1(x - x_P) + y_P \tag{3.65}$$

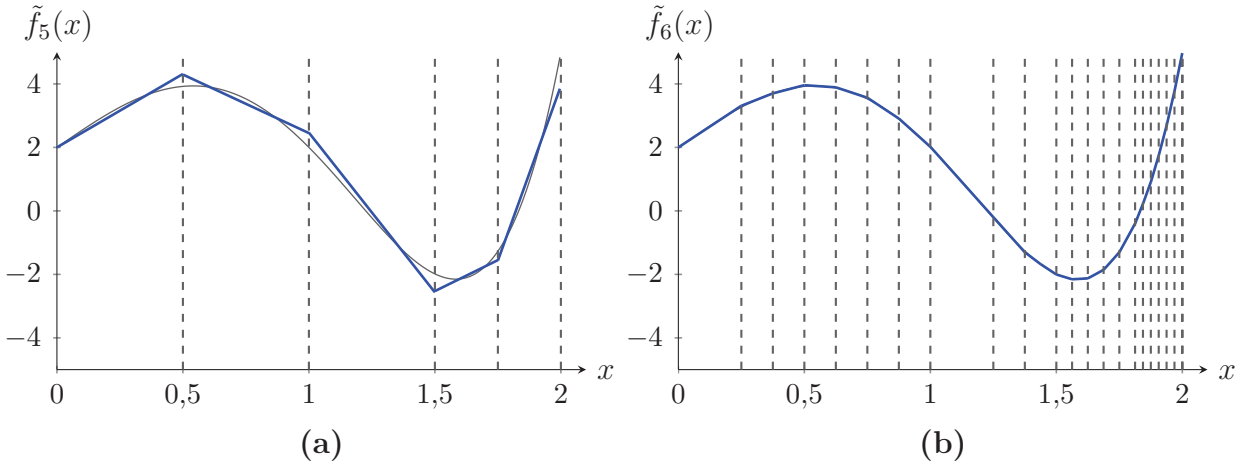


Abbildung 3.10: Beispiel ALFA-basierter Approximationen ohne Funktionssprünge in Abhängigkeit vom Quantisierungsfaktor und mae mit **(a)** $\text{mae}_{ALFA} = 1$, QF$= 3$ und **(b)** $\text{mae}_{ALFA} = 0,05$, QF$= 3$. Im Gegensatz zu der Approximation in Abb. 3.9 ergibt QF$= 1$ keine praktikablen Ergebnisse.

kann durch Minimierung der Steigung nach

$$a_{1,i} = \arg\min_{\hat{a}_{1,i}} \left(||\mathbf{f}_i - (\hat{a}_{1,i}(\mathbf{x}_i - x_{P,i}) + y_{P,i})||_1 \right) \ , \tag{3.66}$$

mit $\mathbf{f}_i$ bzw. $\mathbf{x}_i$ als Teilfunktion bzw. Definitionsbereich im i-ten Segment, eine Näherung der Steigung erstellt werden. Der Achsenabschnittskoeffizient $a_{0,i}$ lässt sich durch $\mathbf{x}_i = \mathbf{0}$ bestimmen. Die resultierende Funktionsapproximation wird, wie bereits beschrieben, durch Auswertung der Rechengenauigkeit bewertet und ggf. geteilt. Im Vergleich zum Standardverfahren ist mit einer höheren Anzahl an Segmenten zu rechnen, da die Vorgabe eines Wertepaares in Gl. (3.65) die Anzahl an verwendbaren Approximationen deutlich einschränkt. Zwei graphische Beispiele der Funktionsapproximation ohne Sprungstellen sind in Abb. 3.10 dargestellt.

Nach der Ermittlung sämtlicher Teilfunktionen müssen die Parameter $\boldsymbol{\kappa}$, λ_j und $\mathbf{a}_0$, die direkt aus den entstandenen Gleichungen folgen, an die ALFA HDL-Erzeugung weitergegeben werden, wofür sie in eine Textdatei geschrieben werden. Ein Beispiel hierfür findet sich in Anhang A.2.

3.7.2 HDL-Erzeugung

Nach der Bestimmung der Parameter der Funktionsapproximation wird eine äquivalente HDL-Beschreibung erzeugt. Dies geschieht unter Verwendung speziell angelegter Dateien, sogenannter *String-Templates*, die die Struktur der zu entwerfenden HDL-Beschreibung bereits beinhalten [85]. Zur textbasierten Umsetzung wurde dabei

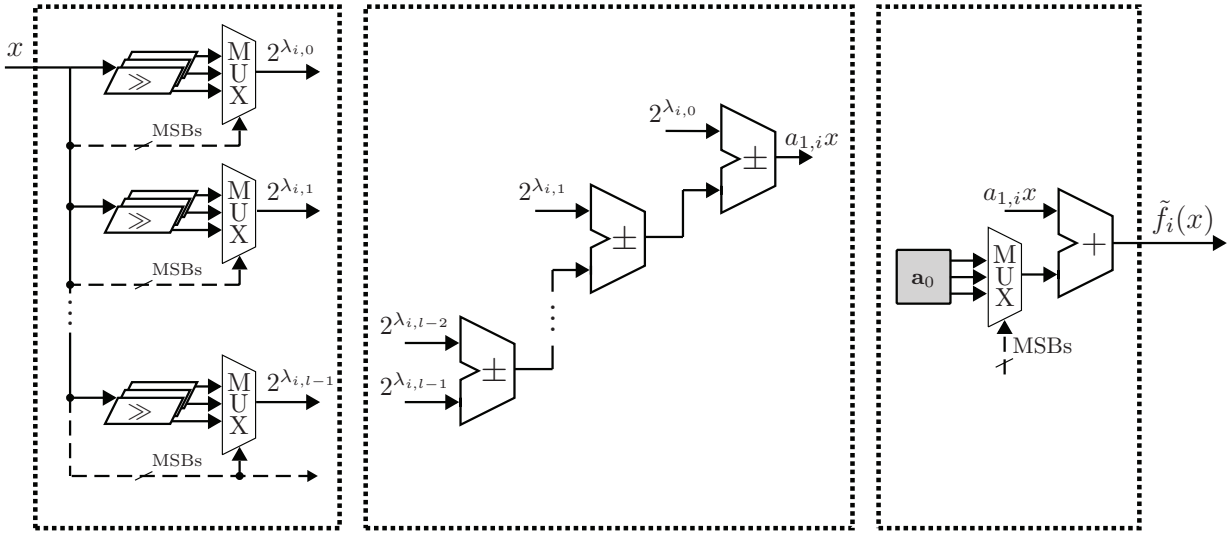


Abbildung 3.11: Aufbau einer durch ALFA erstellten digitalen Schaltung. In **(a)** wird der Koeffizient a_1 gemäß den Ergebnissen der Parameterextraktion bestimmt. Die Schiebeoperatoren sind zur Berechnung einzelner partieller Produkte notwendig. Die MSBs realisieren die in Gl. (3.60) beschriebene Segmentierung. Hiernach folgt in **(b)** die Akkumulation der einzelnen Terme. Die Anzahl der Addierer wird dabei durch den Quantisierungsfaktor festgelegt. In **(c)** wird der Achsenabschnitt a_0 aufsummiert, der sich nach dem aktuellen Segment richtet, weswegen für dessen Auswahl auch die MSBs betrachtet werden müssen.

die Hardware-Entwurfssprache Verilog ausgewählt [51]. Das grundlegende Prinzip der HDL-Erzeugung basiert also auf der Abbildung der extrahierten Parameter aus Kap. 3.7.1 auf äquivalente Hardwarestrukturen, wofür entsprechende Schaltungselemente generisch erzeugt und verdrahtet werden müssen. Der allgemeine Hardwareaufbau ist in Abb. 3.11 gegeben.

Die Zweierpotenzkoeffizienten $\lambda_{i,j}$, die die quantisierte Multiplikation repräsentieren, werden durch Schiebeoperatoren und Addierer umgesetzt. Die Laufvariable j bestimmt dabei die Anzahl der Teilprodukte, die akkumuliert werden. Die Koeffizienten in $\boldsymbol{a}_0$ können direkt als Konstanten übernommen werden. Ihre Hardwareumsetzung geschieht somit zu einem späteren Zeitpunkt im Hardwareentwurf durch Verwendung TIEHI- und TIELO-Gattern, welche direkt aus der Versorgungsspannung bzw. dem Bezugspotential logische Signale erzeugen. Die Abbildung der Segmentierung $\boldsymbol{\kappa}(x)$ geschieht durch kaskadierte Multiplexer, die je nach Länge der betrachteten MSBs ausgewählt werden. Durch die abschließende Verdrahtung sämtlicher Schaltungselemente ergibt sich die Abbildung der spezifizierten Funktionsapproximation.

3.8 Zusammenfassung

Dieses Kapitel behandelt die Herausforderungen der hardwarebasierten Berechnung elementarer Funktionen sowie zugehörige Lösungsansätze. Die automatische lineare Funktionsapproximation (ALFA) stellt dabei ein spezielles Verfahren dar, das durch Kombination und Automatisierung herkömmlicher Ansätze einen Syntheseschritt speziell zur näherungsweisen Abbildung elementarer Funktionen beschreibt. Ausgehend hiervon werden in den folgenden zwei Kapiteln unterschiedliche Anwendungsfälle betrachtet, bei denen ALFA-basierte Signalverarbeitung zum Einsatz kommt. Dabei ist auch die in Kap. 3.1.3 vorgestellte logarithmische Zahlendarstellung von übergeordneter Bedeutung, da diese für die Berechnung einiger elementarer Funktionen einen deutlich reduzierten Signalverarbeitungsaufwand aufweist.

4 Prädiktion von Messdaten im Container

Die Überwachung von Umweltparametern basierend auf drahtlosen Sensornetzen findet heutzutage in vielen unterschiedlichen Bereichen Anwendung [112]. So lassen sich durch die Auswertung von Messdaten Erkenntnisse über den aktuellen Zustand z.B. der Temperatur oder Luftfeuchtigkeit gewinnen, mit Hilfe derer gezielt Folgeprozesse angestoßen werden können. Ein wichtiges Anwendungsfeld hierfür findet sich im Bereich Logistik beim Transport verderblicher Waren und Güter, bei dem nur optimale Umgebungsbedingungen maximale Haltbarkeit der Ware gewährleisten [68]. Zudem erlaubt die präzise Rekonstruktion des Messdatenverlaufs eine aussagekräftige Evaluation der Umweltparameter im Nachhinein. Aufgrund der Menge und Vielzahl unterschiedlicher zu transportierender Waren sowie der Komplexität der Ansteuerung sind diese Prozesse heutzutage i.d.R. selbststeuernd. Die Aufgaben werden dabei dezentral und autonom direkt an einzelnen Transporteinheiten, z.B. Warenkisten oder Containern, durch den Einsatz von Sensorknoten bearbeitet.

Ein kritischer Aspekt der sensorbasierten Überwachung von Umweltparametern im Container ist der Energieverbrauch [54]. So sind Sensorknoten i.d.R. batteriebetrieben, was bei einer langen Transportdauer und/oder hohem Energieverbrauch zu Ausfällen führen kann und somit die flächendeckende Überwachung gefährdet. Die kritische Komponente stellt dabei der Transceiver dar, der bei hoher Auslastung die Batterielaufzeit signifikant reduziert [94]. Um diesen Einfluss zu minimieren, hat sich der Einsatz von sogenannten Prädiktoren bewährt, die für die Vorhersage des Messkurvenverlaufs eingesetzt werden. Jedoch kann ihr Einsatz mit hohem Rechenaufwand einhergehen, was sich ebenfalls negativ auf den Energieverbrauch auswirkt (siehe Kap. 4.2.4). Dieses Kapitel behandelt daher die Realisierung von Prädiktoren mit hoher Genauigkeit und geringem Rechenaufwand für die energieeffiziente, sensorgestützte Überwachung verderblicher Waren im Container.

Nach einem Überblick über grundlegende Aspekte drahtloser Sensornetze werden Algorithmen unterschiedlicher Prädiktoren vorgestellt. Der Schwerpunkt liegt dabei auf künstlichen neuronalen Netzen (KNN), die sich durch anwendungsspezifische Auswertung als geeignete Wahl herausstellen. Hiernach werden sowohl Programmierungs- als auch Implementierungsansätze unter Berücksichtigung der in Kap. 3 angesprochenen

Methoden zur performanten Umsetzung elementarer Funktionen für KNN-Prädiktoren betrachtet. Die Evaluation der untersuchten Ansätze schließt dieses Kapitel ab.

4.1 Drahtlose Sensornetze

Drahtlose Sensornetze können als verteilte Systeme angesehen werden, bei denen die Verarbeitung von Prozessen auf einzelne, untereinander vernetzte Komponenten aufgeteilt wird [117]. Aufgrund ihres speziellen Aufbaus lassen sich einige Besonderheiten erkennen [2]: So können beispielsweise Korrelationseffekte in Messdaten bei räumlich benachbarten Sensorknoten auftreten, die z.B. zur Fehlererkennung eingesetzt werden. Zudem kann in einigen Fällen der Verlust einzelner Messdaten bei genügend hoher Messfrequenz toleriert werden, da sich z.B. bei Umweltparametern die Werte häufig nur langsam verändern. Durch die Installation vieler flächendeckend verteilter und untereinander vernetzter Sensoren ist es möglich, Messungen mit hoher Präzision durchzuführen. Um eine autonome Verarbeitung umzusetzen, müssen Sensornetze sich selbst konfigurieren, also an den aktuellen Zustand des Netzwerks oder die zu überwachende Umgebung anpassen.

Im Bereich sensorgestützter Überwachung von Messdaten für Transport und Logistik lassen sich unterschiedliche Anforderungen, z.B. Zuverlässigkeit, Sicherheit, Skalierbarkeit oder Energieeffizienz, erkennen [6] [60]. Letzt genanntem Aspekt ist dabei übergeordnete Bedeutung zuzuordnen, da er die anderen aufgeführten Punkte maßgeblich beeinflusst. Wird eine Verbesserung der Energieeffizienz erzielt, ist dies gleichbedeutend mit höheren Kapazitäten z.B. für komplexere Programme bezüglich Sicherheitsprotokollen oder für aufwändigere Sensorik. Durch längere Betriebsdauer einzelner Knoten kann zudem die Robustheit eines Systems verbessert werden, da sich die Ausfallwahrscheinlichkeit reduziert. Die Optimierung der Energieeffizienz stellt somit einen essenziell wichtigen Aspekt drahtloser Sensornetze dar und wird daher, nach einem Überblick über gängige Architekturen und Sensorknoten, in Kap. 4.1.3 anwendungsspezifisch, also für den Einsatz im Container, näher betrachtet.

4.1.1 Architektur

Die Architektur drahtloser Sensornetze wird durch unterschiedliche Aspekte wie Topologie und Hierarchie des Sensornetzwerks, aber z.B. auch die räumliche Lage oder die Anzahl der aktiven Sensorknoten, bestimmt. Allgemein findet der Datenaustausch zwischen einem Sende- und einem Empfangsmodul, auch Datenquelle und -senke genannt, statt. Zudem sind verschiedene physikalische Übertragungskanäle, z.B. RF-basierte, optische oder induktive Kommunikation, zur Übertragung denkbar [57]. Für die Er-

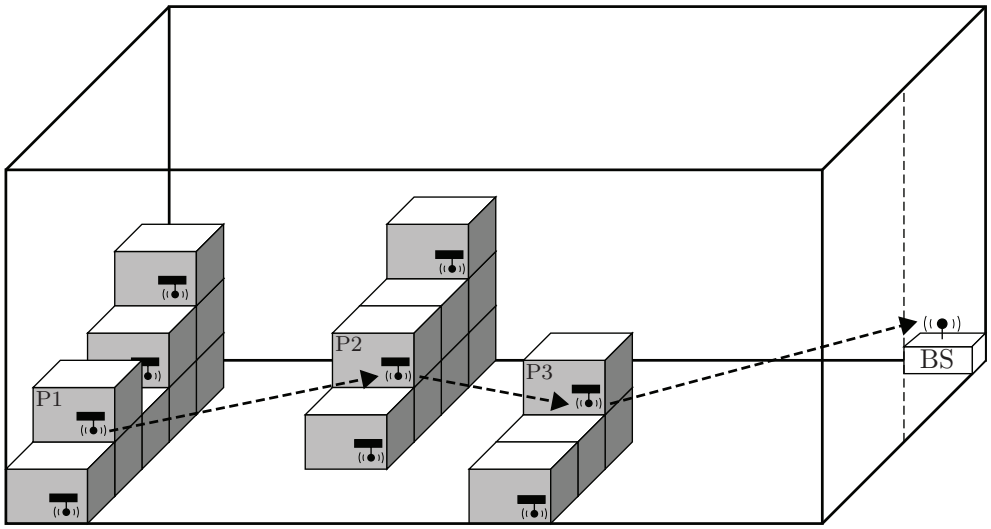
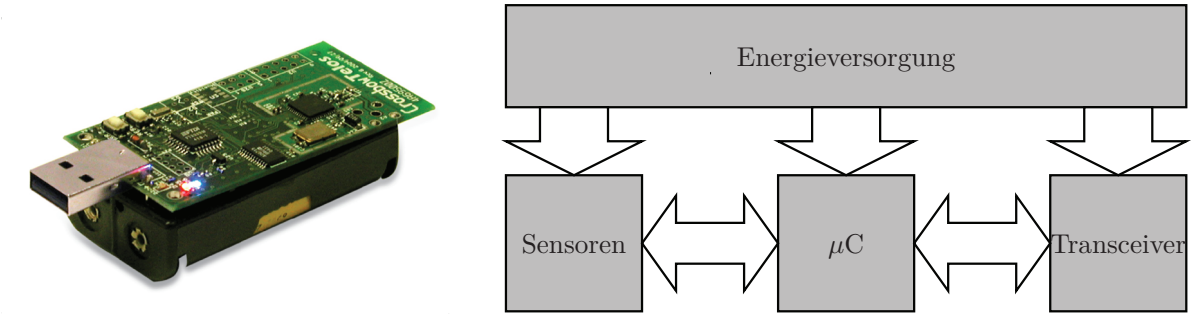


Abbildung 4.1: Beispiel eines drahtlosen Sensornetzes im Container. Jedes transportierte Paket ist mit einem batteriebetriebenen Sensorknoten ausgestattet, der für die Messung der Umweltparameter herangezogen werden kann. Für die Übertragung von Paket 1 (P1) zur Basisstation (BS) ist eine denkbare Multihop-Verbindung mittels der gestrichelten Pfeile skizziert.

mittlung der Umweltparameter im Container wird i.d.R. RF-basierte Kommunikation betrachtet, weswegen in dieser Arbeit nur dieser Fall behandelt wird.

Nach [57] lassen sich Singlehop, Multihop und Cluster als gängige Realisierungen von Netzwerkarchitekturen drahtloser Sensornetze identifizieren. Singlehop-Verbindungen stellen dabei die einfachste Realisierung zum Datenaustausch dar: Hierbei werden alle Datensenken in Reichweite direkt von der Datenquelle adressiert, wodurch der zusätzliche Signalverarbeitungsaufwand minimal gehalten wird. Jedoch können Umgebungseigenschaften oder Hindernisse den Funkverkehr stören oder gar vollständig unmöglich machen, was die Verteilung der Sensorknoten entsprechend limitiert. Um diese Einschränkungen zu umgehen, werden bei Multihop große Distanzen mit Hilfe einer oder mehrerer Zwischenstationen überbrückt. Zudem geht mit diesem Ansatz eine Reduktion des Verbrauches an Übertragungsenergie einher, da hierbei die Distanz in kürzere Unterdistanzen aufgeteilt wird. Bei der Unterteilung von Sensorknoten in Cluster findet eine Hierarchisierung statt, d.h. die Sensorknoten werden ausgehend von ihrer räumlichen Lage gruppiert und einer der Knoten, der so genannte *Clusterhead* ist für die Ansteuerung, z.B. zur Konfiguration der übrigen *Clustermember*, zuständig. So können einzelne Knoten bei kritischem Batteriestand oder aufgrund von Korrelationseffekten vom *Clusterhead* abgeschaltet werden. Im Rahmen dieser Arbeit wird auf Multihops zurückgegriffen, die zur Auswertung der behandelten Prädiktoren vollauf genügen.



(a) (b)

Abbildung 4.2: **(a)** Abbildung eines Moteiv TelosB-Sensorknotens [81] sowie **(b)** der strukturelle Aufbau der zugehörigen Komponenten.

4.1.2 Sensorknoten

Neben der Netzwerkarchitektur beeinflusst auch der eigentliche Aufbau des Sensorknotens die Arbeitsweise drahtloser Sensornetze. Für die in dieser Arbeit verwendeten TelosB-Sensorknoten der Firma Moteiv [81] lassen sich vier grundlegend unterschiedliche Komponenten erkennen, deren Funktionen im Folgenden vorgestellt werden.

Sensorik Die Sensorik realisiert die Messung von Umweltdaten und stellt somit die direkte Schnittstelle zwischen der Elektronik und der Umgebung dar [57]. Da sich der Aufwand der Aufbereitung bedingt durch die verschiedenen physikalischen Verfahren zur Bestimmung von Messgrößen sehr stark unterscheidet, treten auch hohe Variationen bei unterschiedlichen Sensoren hinsichtlich Größe, Messgenauigkeit, Geschwindigkeit oder Energieverbrauch auf [94]. Des Weiteren existieren passive und aktive Sensoren, die sich prinzipiell dahingehend unterscheiden, ob für die Datenmessung die Umgebung manipuliert bzw. stimuliert werden muss. Zudem ist die Energieaufnahme passiver Sensoren i.d.R. vernachlässigbar gering [57]. Im Rahmen dieser Arbeit soll der Einsatz von Prädiktoren ausschließlich anhand von Temperaturverläufen betrachtet werden, wofür der TelosB standardmäßig mit entsprechender Sensorik ausgestattet ist.

Kommunikationseinheit Die Kommunikationseinheit, auch Transceiver genannt, ist für das Versenden und Empfangen von Daten zwischen Sensorknoten zuständig und bildet somit die Schnittstelle zum Funknetzwerk des drahtlosen Sensornetzes. Transceiver sind heutzutage für eine Vielzahl von Übertragungsarten erhältlich. Zudem vereinen sie dabei notwendige Aufgaben der Signalvor- und -aufbereitung, u.a. Modulation, Demodulation, Filterung oder Verstärkung. Der TelosB ist mit dem CC2420 Modul, einem 2,5 GHz IEEE 802.15.4 kompatiblen RF-Transceiver, ausgestattet.

Signalverarbeitungseinheit Die Signalverarbeitungseinheit ist für sämtliche Aufgaben zur Aufbereitung und Auswertung der Messdaten zuständig. Darüber hinaus werden die angeschlossenen Hardwarekomponenten hier angesteuert und konfiguriert. Generell ist es sinnvoll die Signalverarbeitungseinheit in zwei Komponenten zu unterteilen: den Datenspeicher und den Controller. Ersterer ist zumeist durch gängige nicht-flüchtige Speicherbausteine realisiert, z.B. Flash-Module. Beim Controller wird i.d.R. auf integrierte digitale Schaltungen zurückgegriffen. Um den Stromverbrauch möglichst gering zu halten, werden zumeist Mikrocontroller mit geringem Energieverbrauch verwendet, deren CPU nur über einen einfachen Instruktionssatz verfügt, d.h. aufwändige Algorithmen können hiermit nur mit hohem Aufwand umgesetzt werden. Der TelosB besitzt als Signalverarbeitungseinheit den MSP430 Mikrocontroller von Texas Instruments, der sich durch geringe Stromaufnahme auszeichnet [118].

Energieversorgungseinheit Die Energieversorgungseinheit ist, im Gegensatz zu den vorangehend vorgestellten Modulen, nicht direkt an der Verarbeitung der Nutzdaten beteiligt. Vielmehr stellt sie den übrigen Modulen die benötigte elektrische Energie zur Verfügung. Da Sensorknoten energieautark arbeiten, ist die Energieversorgung essenziell für die Betriebsdauer des Sensorknotens. Neben dem Einsatz traditioneller Batterien können auch weitere Ansätze zu Energieerzeugung, z.B. *Energy-Harvesting*, in Betracht gezogen werden. Da sich allerdings im Bereich der Logistik noch keine zufriedenstellenden Ergebnisse erzielen lassen, werden diese Verfahren hier nicht weiter betrachtet. Auch für den TelosB wird im Rahmen dieser Arbeit auf zwei Standard AA-Batterien zurückgegriffen.

4.1.3 Energieverbrauch im Container

Wie bereits in Kap. 4.1 erwähnt ist die Energieeffizienz des einzelnen Sensorknotens von essenzieller Bedeutung für Betrieb und Qualität des gesamten Sensornetzes. Allgemein ist es möglich, den Energieverbrauch der vorab genannten Komponenten des Sensorknotens anhand mathematischer Modelle für die Containerumgebung abzuschätzen [6]. Durch Angabe spezifizierter Werte einzelner Komponenten kann der Gesamtenergieverbrauch berechnet werden. Zur Energiemodellierung kann zum Einen auf die in [81] angegebenen Werte für die Leistungsaufnahme zurückgegriffen werden. Zum Anderen muss anhand eines gängigen Betriebsszenarios die Laufzeit jeder Komponente bestimmt werden. So wird im herkömmlichen Fall ein festes Zeitintervall zwischen zwei Messpunkten festgelegt. Damit keine kritischen Ereignisse oder Veränderungen unerkannt bleiben, muss dieses jedoch möglichst klein gewählt werden. Der gemessene Wert wird zur weiteren Auswertung an die Basisstation gesendet. Zwischen einzelnen Messungen wechselt der Mikrocontroller zur Minimierung des Energieverbrauchs in einen passiven *Sleep*-Modus; der Transceiver bleibt nach dem Sendevorgang für 500 ms empfangsbe-

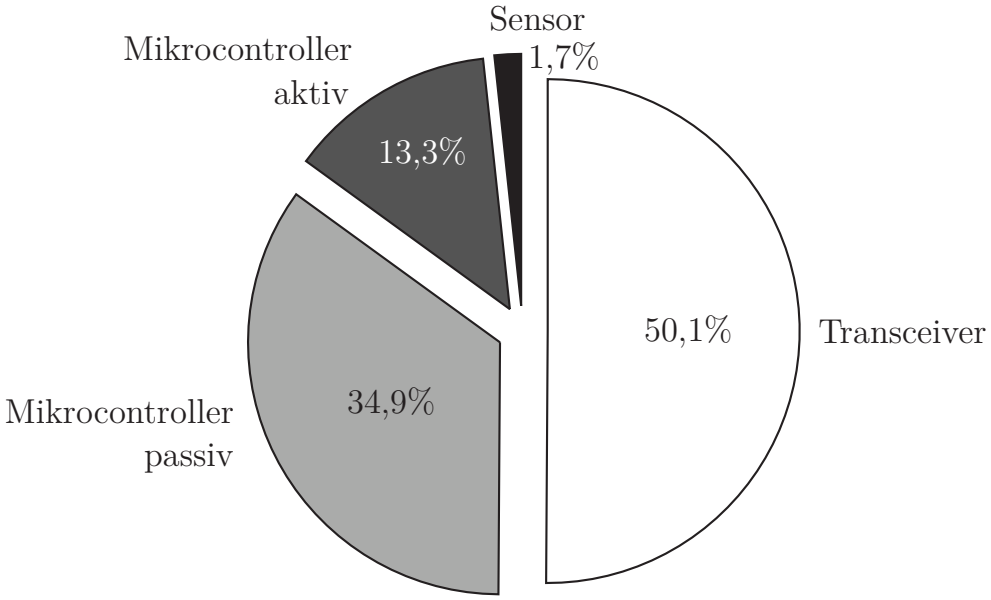


Abbildung 4.3: Relativer, komponentenspezifischer Energieverbrauch eines Sensorknotens pro Minute im Container nach [6], der die Transceivereinheit (Sendeleistung $P_{Out} = 1$ mW) deutlich als größten Verbraucher ausweist.

reit und wird dann deaktiviert. Der relative Energieverbrauch für dieses Betriebszenario mit einem festen Zeitintervall von fünf Minuten zwischen zwei Messpunkten nach [6] ist in Abb. 4.3 visualisiert.

Die Auswertung des relativen Energieverbrauchs ergibt, dass die Transceiver-Aktivität die anderen Komponenten deutlich dominiert. Somit hat die Minimierung des Energieverbrauchs des Transceivers den größten Einfluss auf die Steigerung der Energieeffizienz. Im Rahmen dieser Arbeit wird hierfür der Einsatz von sogenannten Prädiktoren betrachtet, die durch zusätzlichen Signalverarbeitungsaufwand den Transceiver-Energieverbrauch senken. Das grundlegende Prinzip sowie gängige Ansätze zur Prädiktion von Messdaten werden nachfolgend beschrieben.

4.2 Prädiktion von Messdaten

Wie vorangehend dargestellt geht an jedem Sensorknoten eine Messung direkt mit einem Sendevorgang einher, weswegen eine Expansion der bislang statisch angenommenen Messintervalle gleichbedeutend mit einer Reduktion der Transceiver-Aktivität ist und in höherer Energieeffizienz und Betriebsdauer resultiert. Um dies zu erreichen, werden im Rahmen dieser Arbeit Prädiktoren betrachtet, die allgemein zur Vorhersage von unbekannten Variablen anhand gegebener Werte herangezogen werden können [108]. Bezogen auf den Kontext der drahtlosen Sensornetze in der Logistik lässt sich ein Prädiktor als Anwendung ansehen, die anhand vorangehend ermittelter Messdaten die

Verhaltensweise, also die Entwicklung von Umgebungsparametern, vorhersagt. Durch Auswertung des so ermittelten Werts ist es möglich den Zeitpunkt zu bestimmen, an dem die nächste Messung durchgeführt werden soll. So können, wenn keine kritischen äußeren Einflüsse auftreten, die Messintervalle deutlich vergrößert werden, was zu einer Reduktion der Transceiver-Aktivität führt.

Allgemein lassen sich Prädiktoren als Ein- und Mehrschrittverfahren klassifizieren, die sich in der Anzahl an betrachteten Messungen unterscheiden: So werden bei Einschrittverfahren i.d.R. nur Daten vom Zeitpunkt der letzten Messung an betrachtet; bei Mehrschrittverfahren muss eine Reihe an Messungen vorliegen. Nach [53] liefern Einschrittverfahren vergleichsweise schlechte Ergebnisse bei höherem Rechenaufwand, weswegen in dieser Arbeit ausschließlich Mehrschrittverfahren untersucht werden.

Zur Umsetzung des Prädiktors lassen sich unterschiedliche Realisierungen angeben, denen jeweils verschiedene mathematische Ansätze zugrunde liegen. Zudem können sowohl die Qualität der Vorhersage, d.h. wie schnell der zugehörige Algorithmus sich auf veränderte Umgebungsbedingungen einstellt, als auch die Komplexität der Berechnungen variieren. So führen aufwändige Algorithmen zu signifikant höherer Aktivität des Mikrocontrollers, was sich auch negativ auf den Gesamtenergieverbrauch auswirkt. Je geringer somit die Rechenlast des Mikrocontrollers, desto besser die Energieeffizienz. Anhand dieser Überlegungen können die drei Aspekte

- Rechenzeit des Prädiktors,
- Sendeaktivität das Transceivers / Variabilität der Messintervalle und
- Prädiktionsgenauigkeit

identifiziert werden, die als Bewertungsgrundlage für die im weiteren Verlauf betrachteten Auswertungen herangezogen werden.

Im Folgenden werden drei verschiedene Prädiktoren, die auf unterschiedlichen mathematischen Verfahren basieren, vorgestellt und verglichen. Wie in Kap. 4 erwähnt liegt der Fokus dabei aufgrund der übergeordneten Relevanz im Kontext dieser Arbeit auf KNN-basierten Prädiktoren. Anschießend wird die Prädiktionsgenauigkeit der drei Ansätze anhand eines typischen Temperaturverlaufs im Container evaluiert (siehe Abb. 4.7), wodurch die Verwendung des KNN-Prädiktors begründet werden kann.

4.2.1 Methode der kleinsten Quadrate

Das grundlegende Prinzip der KQ-Methode zur polynomialen Regression wurde in Kap. 3.5.1 erläutert. Im Bereich der Prädiktion wird anhand vorangehend gemessener Daten ein approximierter Funktionsverlauf bestimmt. Jeder neue Messpunkt soll dabei als Stützpunkt verwendet werden, weswegen für Polynome hohen Grades keine annehmbare Laufzeit zur Berechnung zu erwarten ist. Es sollen daher zur Prädiktion von

Messdaten ausschließlich lineare Lösungen betrachtet werden. Gl. (3.50) vereinfacht sich dadurch zu

$$\begin{pmatrix} \hat{a}_0\tilde{x}_0 + \hat{a}_1 \\ \hat{a}_0\tilde{x}_1 + \hat{a}_1 \\ \vdots \\ \hat{a}_0\tilde{x}_n + \hat{a}_1 \end{pmatrix} = \tilde{\mathbf{X}} \begin{pmatrix} \hat{a}_0 \\ \hat{a}_1 \end{pmatrix} \approx \tilde{\mathbf{y}} \ . \tag{4.1}$$

Die Lösung der KQ-Methode für Gl. (4.1), die also die Koeffizienten des linearen KQ-Prädiktors liefert, ergibt die Rechenvorschrift

$$\begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \frac{\sum\limits_{k=k_0}^{n} \left(\tilde{x}[k] - \overline{x}\right)\left(\tilde{y}[k] - \overline{y}\right)}{\sum\limits_{k=k_0}^{n} \left(\tilde{x}[k] - \overline{x}\right)^2} \begin{pmatrix} 1 \\ -1 \end{pmatrix} + \begin{pmatrix} 0 \\ \overline{y} \end{pmatrix} \tag{4.2}$$

mit $\tilde{y}[k] = \tilde{y}_k$, $\tilde{x}[k] = \tilde{x}_k$ als k-ter Stützpunkt der Messreihe, $\overline{x}$ bzw. $\overline{y}$ als arithmetisches Mittel der $\tilde{x}$- bzw. $\tilde{y}$-Werte und k_0 als Startwert der Folge, der die Anzahl an betrachteten Eingangswerten (im Folgenden $k_0 = 4$) festlegt. Durch Aufstellen der zugehörigen Geradengleichung kann ein Prädiktionswert $y_p = y[k+1]$ ermittelt werden.

4.2.2 *Modified-Adams*-Methode

Die *Modified-Adams*-Methode (MAM) kann grundlegend als Extrapolationsverfahren klassifiziert werden, bei dem ein Funktionsverlauf durch Interpolation vorgegebener Stützstellen bestimmt wird [111]. Ausgehend von der Definition einer gewöhnlichen Differentialgleichung $\frac{\partial}{\partial k} y(x) = f(y, x)$, mit dem Anfangswert $y(0) = y_0$, folgt durch Integration der Lösungsansatz

$$y_P = y[k] + \int_{x[k]}^{x[k+1]} f(x(k), k) dk \ , \tag{4.3}$$

mit $x[k]$ bzw. $y[k]$ als Wert bzw. Ergebnis zum Zeitpunkt k, $f(x[k], k)$ als genäherter Funktionsverlauf zwischen den Messpunkten $(y[k], x[k])$ und $(y[k+1], x[k+1])$ sowie $y_P = y[k+1]$ als Prädiktionswert [53]. Für den MAM-Prädiktor kann das Integral durch verschiedene Interpolationsverfahren genähert werden, wobei im Rahmen dieser Arbeit die Newton-Gregory-Interpolation betrachtet wird. Der Funktionsverlauf zwischen zwei Messpunkten wird durch Bildung von rekursiven dividierten Differenzen berechnet [45]. Hieraus ergibt sich (durch Variation des zeitlichen Bezugspunktes) nach [53] sowohl der Adams-Bashford-Prädiktor

$$y_P^{ab} = y[k] + h \left(\frac{55f[k] - 59f[k-1] + 37f[k-2] - 9f[k-3]}{24} \right) \tag{4.4}$$

als auch der Adams-Moulton-Korrektor

$$y_P^{am} = y[k] + h\left(\frac{9f[k+1] + 19f[k] - 5f[k-1] + f[k-2]}{24}\right) , \tag{4.5}$$

mit h als äquidistante Intervallgröße zwischen zwei Messpunkten für vier benötigte Stützstellen. Durch näherungsweise Bestimmung der benötigten Funktionswerte mit dem Sekantenverfahren

$$f[k] \approx \frac{y[k] - y[k-1]}{h} \tag{4.6}$$

und mit $y[k+1] = y_P^{ab}$ folgt durch Einsetzen von Gl. (4.4) in Gl. (4.5) der vollständige MAM-Prädiktor

$$y_P^{mam} = \frac{(509y[k] - 534y[k-1] + 336y[k-2] - 146y[k-3] + 27y[k-4])}{192} . \tag{4.7}$$

Eine detaillierte Herleitung findet sich u.a. in [53].

Eine Besonderheit des MAM-Prädiktors ist die Variation von Intervallgrößen, da Gl. (4.7) nur für äquidistante Intervalle h zwischen zwei Messpunkten gilt. Dies hat zur Folge, dass eine Veränderung von h zur Laufzeit nur eingeschränkt möglich ist. So ist die Variation der Messintervalle des MAM-Prädiktors nach [53] auf Verdoppelung bzw. Halbierung der aktuellen Intervallgröße beschränkt, was durch einen größeren Satz an Messdaten bzw. Interpolation umgesetzt wird. Die Entscheidung, ob Intervalle vergrößert oder verkleinert werden, wird unter Zuhilfenahme eines Grenz- oder auch Toleranzwerts *tol* getroffen, der je nach Anwendung angepasst werden muss. Es gilt

$$h = \begin{cases} \frac{h}{2} & \text{, wenn } \quad |y[k+1] - y[k]| \geq tol \quad \text{und} \quad h \neq h_{min} \\ 2h & \text{, wenn } \quad |y[k+1] - y[k]| < \frac{tol}{2} \quad \text{und} \quad h \neq h_{max} \\ h & \text{, sonst} \end{cases} , \tag{4.8}$$

mit h_{max} und h_{min} als maximale und minimale Intervallgröße. Diese Einschränkung hat zur Folge, dass der MAM-Prädiktor ggf. nur verzögert auf starke Temperaturschwankungen reagieren kann.

4.2.3 Künstliche neuronale Netze

Künstliche neuronale Netze (KNN) sind ein wichtiger Teilbereich der künstlichen Intelligenz (KI), die sich mit der Verarbeitung von Informationen durch das menschliche Gehirn befasst [41]. Allgemein können KNN-Strukturen dabei als Simulation oder abstrahierter Nachbau des Gehirns interpretiert werden. Als grundlegende Verarbeitungselemente lassen sich Neuronen und Axone erkennen [66], die für Bewertung und Ver-

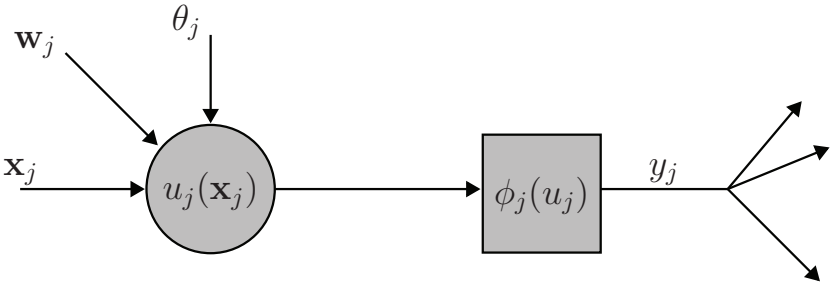


Abbildung 4.4: Einfaches Modell der neuronalen Verbindung j nach [66], bestehend aus einem Axon $u_j(\mathbf{x}_j)$ und einer Aktivierungsfunktion $\phi_j(u_j)$. Jedes i-te Eingangssignal des Eingangsvektors $\mathbf{x}_j$ besitzt eine zugehörige Gewichtung w_{ij}.

netzung von Signalen zuständig sind. Abb. 4.4 zeigt einen beispielhaften elementaren Aufbau. Axone $u(x)$ sind für die Verbindung vorangestellter Eingangs- bzw. Neuronsignale zuständig. Der Einfluss jedes dieser Signale wird zudem über einen zugehörigen Vorfaktor gewichtet. Neurone $\phi(x)$ sind für das Weiterleiten der gewichteten Signale zuständig. Ist der Eingangswert groß genug, wird das Neuron aktiviert und sendet einen Wert. $\phi(x)$ wird daher auch als Aktivierungsfunktion bezeichnet, da erst bei Überschreiten eines spezifischen Schwellwerts θ ein Ergebnis auf den Ausgang gelegt wird. Die tatsächliche Umsetzung einer KNN-Struktur ist durch unterschiedliche Ansätze möglich. Im Rahmen dieser Arbeit soll ausschließlich das Perzeptron Modell nach [97] betrachtet werden. Für eine einschichtige Struktur ergibt sich hierfür die Diskriminantenfunktion

$$u_j(\mathbf{x}) = \sum_i w_{ij} x_j + \theta_j \ , \tag{4.9}$$

mit w als Gewichtung sowie i bzw. j als Eingangs- bzw. Neuronindex.

Zur Anpassung einzelner Neuronen z.B. an wechselnde äußere Einflüsse ist es i.d.R. möglich die Gewichtsfaktoren zur Laufzeit zu verändern. Diese Vorschrift wird u.a. nach [66] durch

$$w_{ij}[k+1] = w_{ij}[k] + \Delta w_{ij}[k] \ , \tag{4.10}$$

zum Zeitpunkt k, beschrieben. Δw_{ij} kann dabei sowohl über statische als auch dynamische Ansätze bestimmt werden. Bei statischer Umsetzung wird i.d.R. die gewünschte Gewichtung manuell eingestellt. Dynamische Ansätze erlauben eine Anpassung zur Laufzeit durch Bewertung des Ergebnisses, z.B. durch Abgleich mit Referenzwerten. Hierdurch ist eine Verbesserung der Signalverarbeitung hinsichtlich Flexibilität erkennbar, was in Anlehnung an die zugrunde liegende Modellierung des Gehirns als Lernen

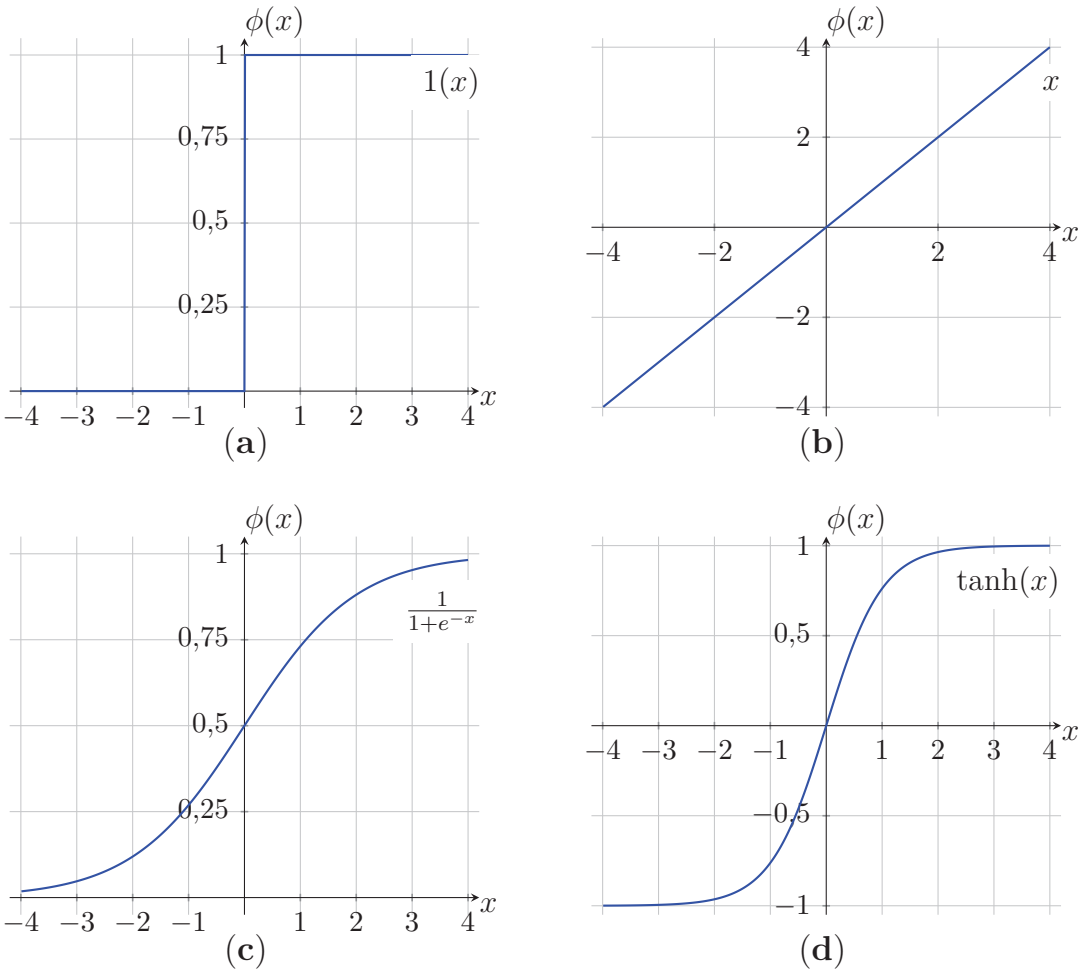


Abbildung 4.5: Verschiedene Umsetzungen der Aktivierungsfunktion $\phi(x)$ durch **(a)** die Heaviside-Funktion und **(b)** die Geradengleichung, die i.d.R. nur für einfache (einschichtige) KNN verwendet werden (siehe Kap. 4.2.3). Für mehrschichtige KNN werden i.d.R. **(c)** die Sigmoidfunktion und **(d)** der hyperbolische Tangens verwendet, die aufgrund ihrer Nichtlinearität aber mit höherem Signalverarbeitungsaufwand einhergehen.

bezeichnet wird. I.d.R. besitzt ein KNN externe Rückführungen, d.h. das Erlernen eines Verhaltens ist durch Vorgabe des gewünschten Ergebnisses von außen möglich.

Durch die bis jetzt angesprochenen Aspekte und Eigenschaften des KNN ist erkennbar, dass es eine Vielzahl an unterschiedlichen Umsetzungsmöglichkeiten gibt. Für das Anwendungsgebiet der Prädiktion von Messdaten hat sich eine Mehrschicht-Perzeptron Struktur (engl.: *multilayer perceptron*, MLP) mit externer Rückführung als geeignet herausgestellt, die im Folgenden beschrieben wird.

Forward Prediction Die *Forward Prediction* ist allgemein der Vorhersage, im Kontext dieser Arbeit also der Prädiktion des nächsten Messwerts, zuzuordnen. Die Re-

chenvorschrift linear gewichteter *Forward Prediction* unter Berücksichtigung von Abb. 4.4 ergibt sich zu

$$y_j = \phi(\mathbf{w}_j \mathbf{x}_j^T) \ , \tag{4.11}$$

mit $\mathbf{w}_j = (w_{0j}, w_{1j}, \ldots, w_{nj}, \theta_j)$ und $\mathbf{x}_j = (x_{j0}, x_{j1}, \ldots, x_{nj}, 1)$. Der Aufbau eines Neurons, also die mathematische Umsetzung der Aktivierungsfunktion $\phi(x)$, hat somit direkten Einfluss auf das Verhalten des KNN, da es, wie vorangehend erwähnt, die Schwelle zur Weiterleitung der gewichteten Signale festlegt. Es existieren daher unterschiedliche Ansätze zur Berechnung von ϕ, die je nach Anwendung auch unterschiedlich gut geeignet sind. Eine Übersicht gängiger Aktivierungsfunktionen ist in Abb. 4.5 dargestellt.

Neben Aufbau und Verhalten der neuronalen Verbindung ist auch die Topologie des KNN von besonderer Bedeutung. Grundsätzlich besteht ein MLP-basiertes KNN aus mehreren Schichten, die ein oder mehrere Neuronen besitzen. Untereinander sind sämtliche Schichten durch

$$\mathbf{x}_k = \begin{pmatrix} \vdots \\ y_j \\ \vdots \end{pmatrix}^T \ , \tag{4.12}$$

mit $\mathbf{x}_k$ als Eingangsvektor einer neuronalen Verbindung, die in der y_j nachfolgenden Schicht liegt, verbunden. Jede Schicht besitzt somit ein Netzwerk zur Verbindung der Ein- und Ausgänge, wobei die Art der Vernetzung prinzipiell frei wählbar ist. Durch Gewichtung jedes Pfades im Netzwerk entsteht eine Gewichtsmatrix $\mathbf{W}$. Hieraus folgt, dass mit einer höheren Anzahl an Schichten i.d.R. bessere Ergebnisse erzielt werden. Bei mehrschichtigen Systemen ist zwischen Eingangs-, Ausgangs- und Zwischenschicht, die auch versteckte Schicht genannt wird, zu unterscheiden. Der Aufbau der in dieser Arbeit verwendeten KNN-Struktur ist in Abb. 4.6 visualisiert. Es werden drei Schichten verwendet, da nach [66] für komplexere Strukturen keine signifikanten Zugewinne erzielt werden.

Backpropagation Die *Backpropagation* ist ein Algorithmus, bei dem die Gewichtsmatrizen dynamisch und durch externe Rückführung trainiert werden. Im Rahmen dieser Arbeit werden die Gewichtsmatrizen dabei zunächst mit Zufallszahlen initialisiert. Durch Simulation, d.h. den Betrieb des KNN-Prädiktors bei Vorgabe realistischer Ein- und Ausgangsdaten, wird das Netzwerk dann vor dem eigentlichen Betrieb konfiguriert. Allgemein wird die *Backpropagation* zumeist nur für mehrschichtige KNN-Strukturen verwendet, deren Verwendung einschichtigen Realisierungen generell vorzuziehen ist, da diese zumeist nur unzureichende Ergebnisse aufweisen [66]. Des Wei-

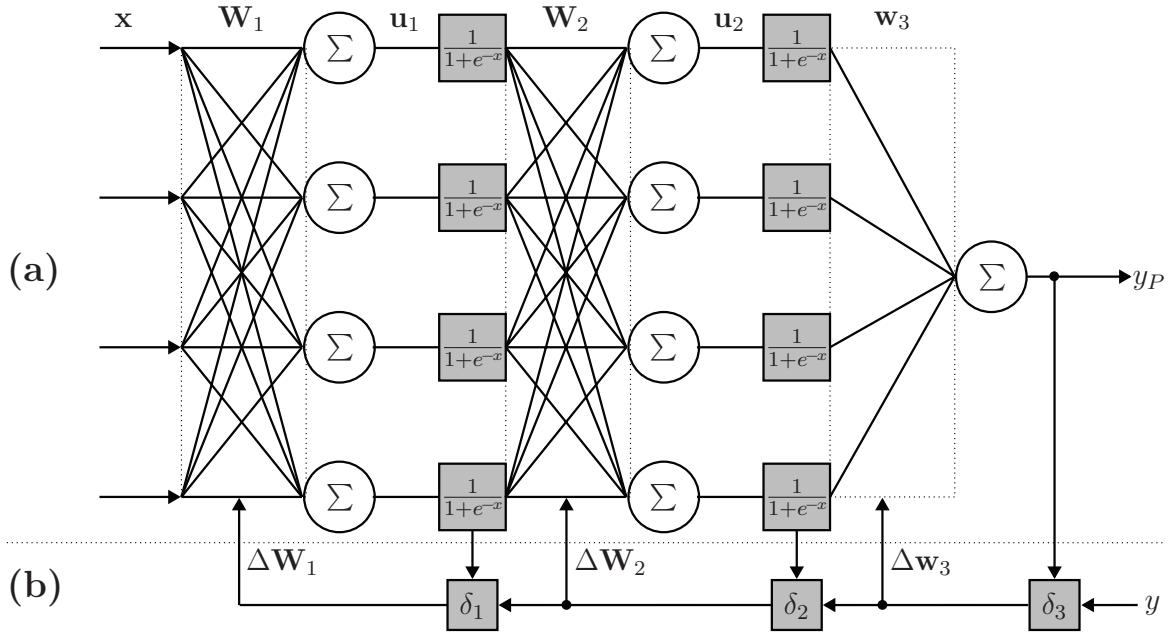


Abbildung 4.6: KNN-Struktur einer **(a)** dreischichtiges MLP mit **(b)** *Backpropagation* und der Sigmoidfunktion als Neuron-Aktivierungsfunktion. In der *Forward Prediction* liegen an den Neuronen gewichtete Werte der vorherigen Schicht bzw. die KNN-Eingänge $\mathbf{x}$ an. Für die Prädiktion werden vorangehende Messdaten als Eingänge verwendet. *Backpropagation* trainiert die Gewichtsmatrizen durch Minimierung der Abweichung der Ausgangssignale (siehe Gl. (4.15)).

teren ist bei MLP-basierten KNN die Benutzung nichtlinearer Aktivierungsfunktionen unabdingbar, da eine Kaskadierung linearer Funktionen immer auf eine einzige lineare Funktion reduziert werden kann.

Bei der *Backpropagation* wird Δw in Gl. (4.10) durch ein Gradientenverfahren, den sogenannten LMS-Algorithmus berechnet, der u.a. nach [41] durch Minimierung der Kostenfunktion

$$E = \frac{1}{2}\varepsilon_j^2 \ , \tag{4.13}$$

mit ε_j als Abweichung des j-ten Neurons, bestimmt werden kann. Die Abweichung der Ausgangsschicht kann durch Ableitung der Kostenfunktion in Gl. (4.13) als

$$\varepsilon = y - y_P \ , \tag{4.14}$$

mit y_P bzw. y als Vorhersage bzw. gemessenem Wert, angegeben werden.

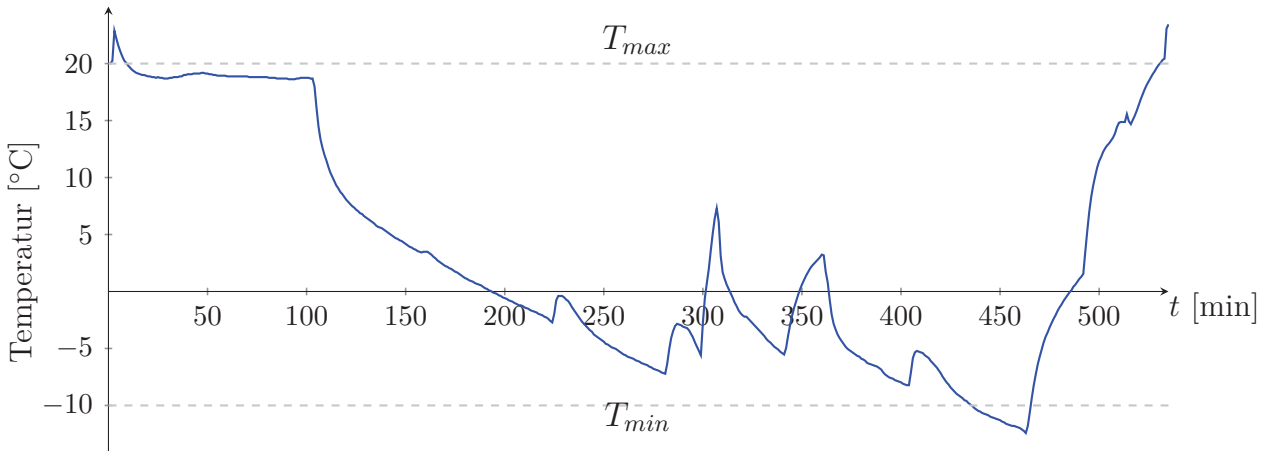


Abbildung 4.7: Verlauf einer im Container gemessenen Temperatur, die wesentliche Betriebsszenarien berücksichtigt, anhand derer, zusammen mit den Temperaturgrenzen $T_{max} = 20°\text{C}$ und $T_{min} = -10°\text{C}$, die unterschiedlichen Prädiktoren in dieser Arbeit evaluiert werden.

Um den minimalen Fehler der Gewichtsfaktoren zu bestimmen, muss E nach w abgeleitet werden. Es ergibt sich

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta \delta_j x_j \ , \tag{4.15}$$

mit η als Lernkoeffizient, der die Konvergenzgeschwindigkeit angibt, und x_j bzw. δ_j als i Eingangsdaten bzw. als lokaler Gradient des j-ten Neurons. Der lokale Gradient ist durch

$$\delta_j = \begin{cases} \frac{\partial}{\partial u_j}\phi(u_j) \cdot \varepsilon & \text{, Gradient der Ausgangsschicht} \\ \frac{\partial}{\partial u_j}\phi(u_j) \cdot \sum\limits_k \delta_k \cdot w_{kj} & \text{, sonst} \end{cases} \ , \tag{4.16}$$

mit k als Laufindex über sämtliche Neuroneneingänge, die mit dem Ausgang des j-ten Neurons verbunden sind, beschrieben. Eine Herleitung von Gl. (4.15) und Gl. (4.16) ist u.a. in [41] beschrieben.

4.2.4 Evaluation

Anhand der beschriebenen theoretischen Grundlagen können nun die im Rahmen dieser Dissertation vorgenommenen Maßnahmen vorgestellt werden, die eine effiziente Umsetzung von Prädiktoren im Containerumfeld erlauben. Hierfür werden die drei vorangehend beschriebenen Algorithmen zunächst allgemein hinsichtlich Laufzeit, Prädiktionsgenauigkeit und Variabilität der Messintervalle untersucht, woraus sich ein geeigneter Ansatz für weitere Optimierungen auswählen lässt.

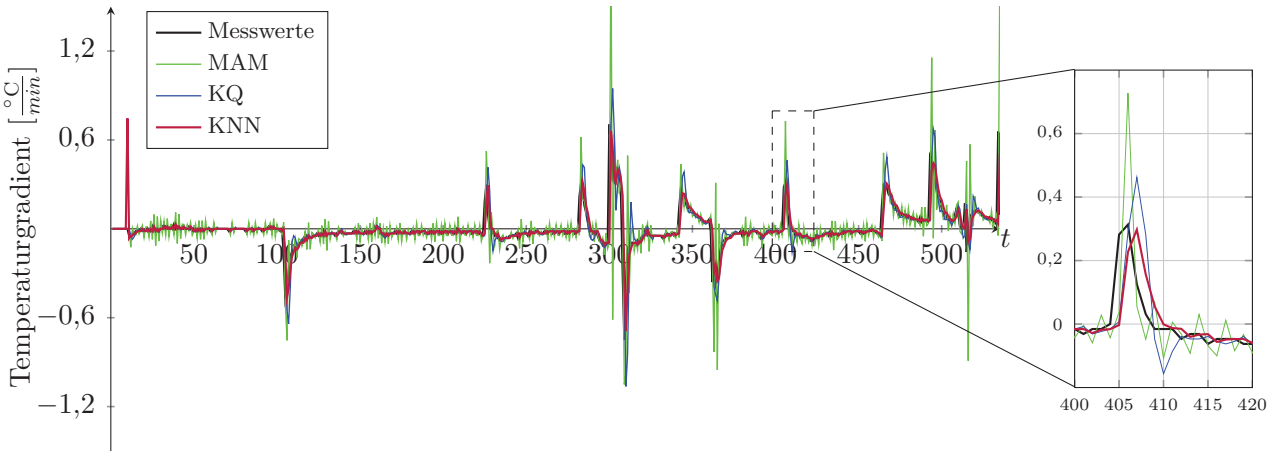


Abbildung 4.8: Steigungsverlauf der in Abb. 4.7 gegebenen Temperaturmessreihe sowie der Verlauf des KQ-, MAM- und KNN-Prädiktors.

Die Laufzeit beeinflusst direkt den Energieverbrauch: Je länger somit der Mikrocontroller für die Abarbeitung des Prädiktor-Programms braucht, desto länger ist die Signalverarbeitungseinheit aktiv, was sich negativ auf die Energieeffizienz auswirkt. Die Prädiktionsgenauigkeit bewertet die Qualität der Vorhersage. So kann für sehr hohe Genauigkeiten u.a. der Temperaturverlauf im Container präziser rekonstruiert werden, wodurch kritische Zustände schneller entdeckt werden können. Die Variabilität der Messintervalle gibt an, wie der Prädiktor auf Änderungen der Umwelt reagiert. Je flexibler der Prädiktor, desto besser (d.h. schneller) ist die Reaktionsfähigkeit, womit i.d.R. eine Senkung des Transceiver-Energieverbrauchs einhergeht.

Zur Analyse sind die Algorithmen als direkte Gleitkomma-Umsetzung auf die TelosB Sensorknoten programmiert. Die Länge der Messreihe ist nach [124] für die KQ-Methode sowie den KNN-Algorithmus auf vier bzw. für MAM auf fünf Werte festgelegt. Das KNN besteht aus drei Schichten zur Gewichtung sowie der Sigmoidfunktion als Aktivierungsfunktion (siehe Abb. 4.5), wie in Abb. 4.6 gegeben. Die Bewertung wird anhand einer Messreihe, die einen in realer Containerumgebung ermittelten Temperaturverlauf angibt, durchgeführt, die in einem Zeitraum von 535 Minuten ein typisches Szenario der im Container auftretenden äußere Einflüsse beschreibt. Die Auflösung der Messkurve beträgt eine Minute. Der vollständige Verlauf ist in Abb. 4.7 gezeigt.

Für die Prädiktion werden des Weiteren Ober- (T_{max}) und Untergrenzen (T_{min}) der Temperatur eingeführt, durch die der Bereich, in der die transportierte Ware keinen Schaden nimmt, festgelegt wird. Die Bestimmung eines Zeitintervalls findet nun wie in [124] anhand des linearen Gradienten, also der Steigung der Temperatur zum aktuellen Zeitpunkt, statt (siehe Abb. 4.8), wodurch der Prädiktor sensibel auf Temperatur-

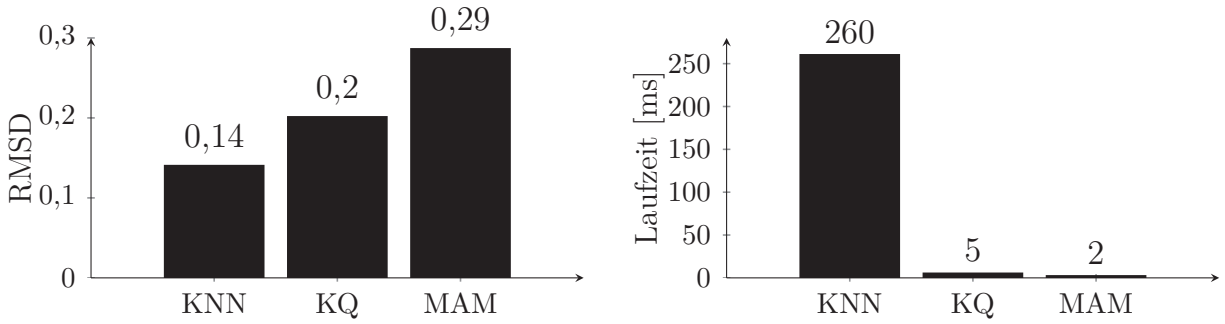


Abbildung 4.9: Evaluation der Prädiktoren hinsichtlich **(a)** Prädiktionsgenauigkeit und **(b)** Laufzeit einer zugehörigen direkten MSP430 Programmierung. Der KNN-Ansatz liefert die höchste Genauigkeit, benötigt jedoch auch die deutlich längste Rechenzeit, die MAM verhält sich exakt gegensätzlich. Die KQ-Methode erreicht in beiden Fällen durchschnittliche Werte.

schwankungen und nicht auf die absoluten Werte reagiert. Die Eingangsdaten ergeben sich zu

$$y[k] = \begin{cases} 0 & \text{, wenn } k = 1 \\ \frac{T[k]-T[k-1]}{t[k]} & \text{, sonst} \end{cases} , \tag{4.17}$$

mit $T[k-1]$ bzw. $T[k]$ als vorheriger bzw. aktuell gemessener Temperaturwert und $t[k]$ als Zeitintervall, gegeben sind. Der so entstehende Gradientenverlauf wird an den Eingang des Prädiktors gelegt. Die Rechenvorschrift lautet hierfür

$$t[k+1] = \begin{cases} \frac{|T[k]-T_{min}|}{s \cdot y_P} & \text{, wenn } y_P < 0 \\ t_{max} & \text{, wenn } y_P = 0 \\ \frac{|T_{max}-T[k]|}{s \cdot y_P} & \text{, wenn } y_P > 0 \end{cases} , \tag{4.18}$$

mit $s = 10$ als Skalierungsfaktor und $t_{max} = 20$ min. Somit kann die Abschätzung des Zeitintervalls durch die aktuell gemessene Temperatur und den aktuellen Prädiktorwert bestimmt werden. Für die gegebenen Messdaten werden Temperaturgrenzen bei $T_{max} = 20$ °C und $T_{min} = -10$ °C gesetzt. Für die KNN-basierte Prädiktion wird zudem der Lernkoeffizient η mit $\eta = 0,8$ angenommen, da dies nach [123] für Messungen im Container bestmögliche Werte erzielt.

Variabilität der Messintervalle Die Länge der Messintervalle kann für KQ- und KNN-Prädiktion frei variiert werden. Der MAM-Prädiktor kann seine Messintervalle, wie in Kap. 4.2.2 beschrieben, nur verdoppeln oder halbieren, was eine deutliche Einschränkung der Flexibilität darstellt.

Laufzeit Zur Messung der Laufzeit wird die interne Peripherie des MSP430 verwendet [118]. Die MAM- und KQ-Prädiktoren erreichen hierbei mit $t_{MAM} = 2$ ms und $t_{MAM} = 5$ ms die besten Ergebnisse. Der KNN-Algorithmus zeigt mit $t_{KNN} = 260$ ms deutlich schlechtere Werte an, was u.a. in der nichtlinearen Aktivierungsfunktion begründet liegt (siehe Kap. 4.2.3).

Prädiktionsgenauigkeit Um die Prädiktoren möglichst gerecht zu vergleichen, werden für die Prädiktionsgenauigkeit, die wie die Variabilität der Messintervalle Einfluss auf die Reaktionsfähigkeit des Prädiktors gibt, feste, äquidistante Messintervalle $t_{fix} = 2$ min vorgegeben. Der Prädiktionsfehler wird durch die Wurzel der mittleren quadratischen Abweichung (engl.: *root mean square derivation*, RMSD) bestimmt, die sich durch

$$RMSD = \sqrt{\frac{\sum_{k=1}^{n} (y_P[k] - y[k])^2}{n}} \,, \tag{4.19}$$

mit y_P bzw. y als Wert der Prädiktion bzw. dem zugehörigen gemessenen Wert zum Zeitpunkt k, berechnen lässt. Die KQ-Prädiktion erzielt bei dieser Auswertung die schlechtesten Werte ($RMSD_{MAM} = 0,029$), gefolgt von KQ ($RMSD_{KQ} = 0,020$). Die besten Ergebnisse lassen sich für den KNN-Prädiktor erkennen ($RMSD_{KNN} = 0,014$). Ein graphischer Überblick der Auswertung ist in Abb. 4.9 dargestellt.

Ausgehend von diesen Ergebnissen soll der KNN-Prädiktor zur Messung von Umweltparametern im Container verwendet werden, da er verglichen mit den anderen Verfahren die besten Ergebnisse hinsichtlich Prädiktionsgenauigkeit und Variabilität der Messintervalle liefert. Um ein effizienteres Verhalten hinsichtlich der Laufzeit zu erreichen, ist es notwendig, den zugrunde liegenden Algorithmus zu vereinfachen, was im Rahmen dieser Arbeit durch Berücksichtigung der in Kap. 3 vorgestellten Verfahren zur näherungsweisen Berechnung elementarer Funktionen geschehen soll. Es werden im Folgenden zunächst softwarebasierte Ansätze untersucht, bevor dann, unter Verwendung der daraus gewonnenen Erkenntnisse, mögliche Hardwareumsetzungen für die KNN-Prädiktoren betrachtet werden.

4.3 Softwarebasierte KNN-Prädiktion

Die Umsetzung der softwarebasierten KNN-Prädiktion basiert auf der Anpassung der KNN-Struktur an die Ressourcen des TelosB-Sensorknotens [105]. Zur Steigerung der Ausführungsgeschwindigkeit und damit, wie bereits erwähnt, zur Erhöhung der Energieeffizienz des KNN-Prädiktors werden sowohl das Zahlenformat als auch algorithmische Terme, die eine übermäßig hohe Rechenkomplexität aufweisen, möglichst hardwarefreundlich umgesetzt.

Für Ersteres wird die ursprünglich verwendete Gleitkomma-Darstellung durch eine Festkomma-Darstellung ersetzt. Die Eingangsdaten werden ins $Q.15$ Zahlenformat skaliert. Die Berechnung der nichtlinearen Aktivierungsfunktion sowie des zugehörigen Gradienten stellt eine hohe Rechenbelastung für den MSP430 dar. Ausgehend von der originalen Sigmoidfunktion

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{4.20}$$

werden zunächst algorithmische Anpassungen betrachtet.

Allgemein wird die Wertemenge der Aktivierungsfunktion durch das vorangehende Ergebnis bestimmt. So ist zunächst jeder einzelne Summand der Gewichtsmatrix durch das $Q.15$ Zahlenformat auf den Wertebereich $-1 \leq x \leq 1 - 2^{-15}$ festgelegt. Zur Vermeidung von Überläufen werden betragsgrößere Zahlen entsprechend auf den jeweiligen Grenzwert gesetzt. Da wie in Kap. 4.2.4 beschrieben jedes Neuron vier Eingangsdaten besitzt, ergibt sich die Wertemenge der Aktivierungsfunktion zunächst zu $-4 \leq x \leq 4 - 2^{-15}$.

Zur Anpassung der Sigmoidfunktion an diese Wertemenge ist es notwendig das Zahlenformat entsprechend abzuändern, was sich durch Modifikation der Gewichtsfunktion aus Gl. (4.9) realisieren lässt. Die Rechenvorschrift lautet dann

$$\tilde{u}_i = \sum_{j=1}^{N} w_{ij} \left(\frac{1}{N} x \right) , \tag{4.21}$$

mit $N = 4$ als Anzahl der gewichteten Terme, woraus sich $\mathrm{ld}(N-1) = 2$ Schiebeoperatoren, die jeweils nach der ersten und zweiten Addition durchgeführt werden, ergeben. Das Ergebnis von Gl. (4.21) liegt somit im $Q2.13$ Zahlenformat vor. Die Aktivierungsfunktion verändert sich unter Berücksichtigung dieser Modifikation zu

$$\tilde{\phi}(\tilde{u}_i) = \frac{1}{1 + e^{-(4\tilde{u}_i)}} \ . \tag{4.22}$$

Neben der Anpassung der *Forward Prediction* an die verwendete Hardware muss auch die *Backpropagation* entsprechend modifiziert werden. Zum Einen ist es, wie vorangehend beschrieben, notwendig, Überläufe, die aufgrund des verwendeten Zahlenformats auftreten können, zu vermeiden. Zum Anderen muss bei *Backpropagation* die Ableitung der Aktivierungsfunktion berechnet werden (siehe Kap. 4.2.3). Dies kann aufgrund der Beschaffenheit der Sigmoidfunktion allgemein in Abhängigkeit von $\phi(x)$ geschehen. Es ergibt sich

$$\frac{\partial}{\partial x}\phi(x) = \phi(x) \cdot (1 - \phi(x)) \ . \tag{4.23}$$

Der in Gl. (4.16) formulierte lokale Gradient δ lässt sich damit für die jeweilige Schicht des MLP als Vektor (oder als Skalar für die Ausgangsschicht) durch

$$\delta_3[k] = -\eta(y[k] - y_P[k]) \tag{4.24}$$
$$\boldsymbol{\delta}_2[k] = (\tilde{\boldsymbol{\phi}}_2[k] \circ (1 - \tilde{\boldsymbol{\phi}}_2[k])) \circ (\delta_3[k] \cdot \mathbf{w}_3[k]) \ , \tag{4.25}$$
$$\boldsymbol{\delta}_1[k] = (\tilde{\boldsymbol{\phi}}_1[k] \circ (1 - \tilde{\boldsymbol{\phi}}_1[k])) \circ (\boldsymbol{\delta}_2[k]^T \cdot \mathbf{W}_2[k]) \tag{4.26}$$

mit y_P und y als Prädiktion und Messwert, $\mathbf{W}_i$ bzw. $\mathbf{w}_i$ als Gewichtsmatrix bzw. -vektor, $\tilde{\boldsymbol{\phi}}_i$ als Ergebnisse der Sigmoidfunktionen der i-ten Schicht und $\circ$ als Hadamard-Operator [25], angeben. Genau wie bei den Gewichtskoeffizienten ist die Differenz $\varepsilon = y - y_P$ auf den Bereich $-1 \leq \varepsilon \leq 1 - 2^{-15}$ begrenzt. Zur weiteren Reduktion des Rechenaufwandes ist zudem der Lernkoeffizient direkt in δ_3 berücksichtigt. Der in Gl. (4.15) gegebene Gradient ergibt eingesetzt in Gl. (4.10) die Rechenvorschrift

$$\mathbf{w}_3[k+1] = \mathbf{w}_3[k] - \delta_3[k] \circ \tilde{\boldsymbol{\phi}}_2[k] \tag{4.27}$$
$$\mathbf{W}_2[k+1] = \mathbf{W}_2[k] - \boldsymbol{\delta}_2[k] \cdot \tilde{\boldsymbol{\phi}}_1[k]^T \ , \tag{4.28}$$
$$\mathbf{W}_1[k+1] = \mathbf{W}_1[k] - \boldsymbol{\delta}_1[k] \cdot \mathbf{x}[k]^T \tag{4.29}$$

mit $\mathbf{x}$ als Eingangsvektor des KNN.

Neben der algorithmischen Festkomma-Umsetzung des KNN-Algorithmus ist die möglichst präzise Approximation der Aktivierungsfunktion ohne signifikante Verluste hinsichtlich Prädiktionsgenauigkeit für die effiziente softwarebasierte Umsetzung elementar. Aufgrund der geringen Kapazität des Datenspeichers ist die Umsetzung z.B. exakter, tabellengestützter Verfahren nicht möglich (siehe Kap. 3.6). Daher sollen sowohl polynomiale Regression (siehe Kap. 3.5.1) als auch uniforme Segmentierung (siehe Kap. 3.6.2) zur Funktionsapproximation im Folgenden untersucht werden. Zur Bewertung findet eine Evaluation der Resultate hinsichtlich Prädiktionsgenauigkeit, Laufzeit und Sendeaktivität anhand der in Abb. 4.7 eingeführten Messreihe statt.

4.3.1 Polynomiale Regression

Zur Umsetzung der Aktivierungsfunktion mittels polynomialer Regression wird der Verlauf der Aktivierungsfunktion durch angenäherte Polynome dargestellt [105]. Hierdurch ergibt sich die Aktivierungsfunktion

$$\tilde{\phi}(x) = \frac{1}{1 + e^{-4x}} \approx \tilde{\phi}_{PA}(x) = \sum_{k=0}^{n} a_k x^k \ , \tag{4.30}$$

mit $\tilde{\phi}_{PA}(x)$ als polynomiale Approximation und a_k als zugehöriger Koeffizient.

Tabelle 4.1: Approximation der Sigmoidfunktion durch polynomiale Regression mit a_k als k-tem Koeffizient und n als Grad des Polynoms. Da die Sigmoidfunktion eine ungerade Funktion ist, werden gerade Polynomterme vernachlässigt.

n	a_9	a_7	a_5	a_3	a_1	a_0
	$[10^{-6}]$	$[10^{-5}]$	$[10^{-3}]$	$[10^{-3}]$	$[10^{0}]$	$[10^{0}]$
9	1,30	-6,60	1,44	-19,46	0,25	0,5
7		-2,20	0,94	-17,43	0,25	0,5
5			0,38	-13,28	0,24	0,5
3				-6,59	0,22	0,5
1					0,15	0,5

Die Bestimmung des Polynoms wird basierend auf dem in Kap. 3.5.1 vorgestellten Regressionsverfahren durchgeführt. Ausgehend von einer den oberen Angaben entsprechenden Funktionsreferenz werden die jeweiligen Koeffizienten für die Polynome des Grades $n = 1$ bis $n = 9$ vorab bestimmt. Da es sich bei der Sigmoidfunktion um eine ungerade Funktion handelt, können gerade Polynomterme bei der Programmierung vernachlässigt werden. Die resultierenden ungeraden Koeffizienten der unterschiedlichen Polynome sind in Tab. 4.1 aufgelistet. Eine graphische Übersicht der approximierten Sigmoidfunktionen sowie der zugehörigen Ableitungen für die *Backpropagation* ist in Anhang A.3 gegeben.

4.3.2 Uniforme Segmentierung

Bei der Verwendung uniformer Segmentierung als Approximationsansatz wird die Aktivierungsfunktion in verschiedene Bereiche aufgeteilt, in denen die zugehörige Teilfunktion approximiert wird [106]. Um den Rechenaufwand hierbei so gering wie möglich zu halten, sollen ausschließlich Polynome ersten Grades, also Geradengleichungen der Form

$$\tilde{\phi}(x) = \frac{1}{1 + e^{-4x}} \approx \tilde{\phi}_{SA}(x) = a_{0,i}^{k} x + a_{1,i}^{k} \ , \tag{4.31}$$

mit i bzw. k als aktuellem Segment bzw. der absoluten Anzahl an Segmenten, verwendet werden. Außerdem muss für einen effizienten Zugriff auf die Segmente die Unterteilung den in Kap. 3.6.2 formulierten Bedingungen entsprechen.

Die Approximation der KNN-Struktur durch uniforme Segmentierung kann in zwei unterschiedliche Phasen aufgeteilt werden: die Initialisierung und die Berechnung zur Laufzeit (siehe Alg. 4.1). Während der Initialisierung werden vor der eigentlichen Prä-

Tabelle 4.2: Approximation der Sigmoidfunktion durch uniforme Segmentierung mit a_0, a_1 als Koeffizienten der Geradengleichung. Die Tabelle listet Koeffizienten für Approximation mit vier ($a^4_{0/1}$), acht ($a^8_{0/1}$) und 16 ($a^{16}_{0/1}$) uniformen Segmenten, sowie das zugehörige Funktionsintervall auf.

Segment	a_1^4	a_0^4	a_1^8	a_0^8	a_1^{16}	a_0^{16}
$(-1, -\frac{7}{8}]$	0,194	0,200	0,116	0,132	0,089	0,107
$(-\frac{7}{8}, -\frac{6}{8}]$					0,143	0,154
$(-\frac{6}{8}, -\frac{5}{8}]$			0,285	0,256	0,225	0,215
$(-\frac{5}{8}, -\frac{4}{8}]$					0,343	0,289
$(-\frac{4}{8}, -\frac{3}{8}]$	0,772	0,476	0,596	0,411	0,502	0,369
$(-\frac{3}{8}, -\frac{2}{8}]$					0,688	0,439
$(-\frac{2}{8}, -\frac{1}{8}]$			0,930	0,496	0,867	0,484
$(-\frac{1}{8}, 0]$					0,983	0,499
$(0, \frac{1}{8}]$	0,772	0,524	0,930	0,504	0,983	0,501
$(\frac{1}{8}, \frac{2}{8}]$					0,867	0,515
$(\frac{2}{8}, \frac{3}{8}]$			0,596	0,589	0,688	0,559
$(\frac{3}{8}, \frac{4}{8}]$					0,502	0,628
$(\frac{4}{8}, \frac{5}{8}]$	0,194	0,800	0,285	0,744	0,343	0,708
$(\frac{5}{8}, \frac{6}{8}]$					0,225	0,782
$(\frac{6}{8}, \frac{7}{8}]$			0,116	0,868	0,143	0,844
$(\frac{7}{8}, 1]$					0,089	0,891

diktion die Koeffizienten a_0, a_1 der approximierten Geraden fest im Datenspeicher hinterlegt. Die Anordnung ist dabei von der Lage des zugehörigen Segments abhängig. Zur Laufzeit findet die eigentliche Berechnung des approximierten KNN-Algorithmus statt. Für die Aktivierungsfunktion werden dabei m MSBs des Eingangsoperanden mit $m = \lfloor \mathrm{ld(k)} \rfloor$ betrachtet. Durch eine sortierte Angabe der Koeffizienten in einem entsprechenden Array während der Initialisierung können die notwendigen Koeffizienten durch Schiebeoperationen ermittelt werden. Die anschließende näherungsweise Berechnung ist durch Multiplikation und Addition möglich (siehe Gl. (4.31)).

Für die Umsetzung werden vier, acht und 16 uniforme Segmente betrachtet. Auch für diesen Approximationsansatz werden die Koeffizienten a_0, a_1 mittels KQ-Methode vorab bestimmt. Die Koeffizienten der approximierten Aktivierungsfunktionen sind in Tab. 4.2 aufgelistet.

4.3.3 Evaluation

Zur Auswertung wird ein TelosB-Sensorknoten mit den unterschiedlichen Funktionsapproximationen unter Berücksichtigung der zuvor angesprochenen Festkomma-Anpassungen programmiert. Als Referenz dient ein KNN-Prädiktor mit unmodifizierter Aktivierungsfunktion. Die Berechnung der Sigmoidfunktion findet hierbei in Gleitkomma-Zahlendarstellung statt, wofür innerhalb des Algorithmus zusätzlich entsprechende Umwandlungen (engl.: *cast*) zur Festkomma-Darstellung berücksichtigt werden müssen. Zur Bewertung werden wie in Kap. 4.2.4 Prädiktionsgenauigkeit, Laufzeit und Sendeaktivität, also die Anzahl an Sendevorgängen des Transceivers, herangezogen. Zur Berechnung der Länge des nächsten Messintervalls wird das Ergebnis des Prädiktors in Gl. (4.18) eingesetzt. Für die Stimulation wird die in Abb. 4.7 gegebene Temperaturmessreihe verwendet. Darüber hinaus werden sowohl drei- als auch zweischichtige KNN-Strukturen untersucht, wobei Letztere nur aus einer Eingangs- und einer Ausgangsschicht bestehen. Ein graphischer Überblick der Ergebnisse ist in Abb. 4.10 gezeigt.

Laufzeit Die Ermittlung der Laufzeit wird durch die internen Zähler des MSP430 realisiert. Sowohl für zwei als auch drei KNN-Schichten benötigt die lineare Umsetzung die geringste Rechenzeit, die direkte Umsetzung der Sigmoidfunktion die meiste. Die uniformen Umsetzungen sind nur unwesentlich langsamer als die linearen Approximationen. Die polynomialen Ansätze liegen zwischen der uniformen und der direkten Programmierung, wobei die Laufzeit mit dem Grad der Näherung ansteigt.

Sendeaktivität Bei Sendeaktivität erzielt die direkte Umsetzung die besten Ergebnisse. Bei den approximativen Ansätzen ist bei zweischichtiger KNN-Struktur für hö-

Algorithmus 4.1: Approximation der Sigmoidfunktion für vier uniforme Segmente

```
// Initialisierung
const short k = 4; // Anzahl Segmente
const short shr = 14; // 16 − ld(k))
// Koeffizienten
const int[k] a0 = {17170, 26214, 6554, 15598}; // a_0^4 · 2^15
const int[k] a1 = {25297, 6357, 6357, 25297}; // a_1^4 · 2^15
⋮
// Laufzeit
int getUniformApproximation(int x) {
  return (a1 [x » shr] · x + a0[x » shr]);
}
```

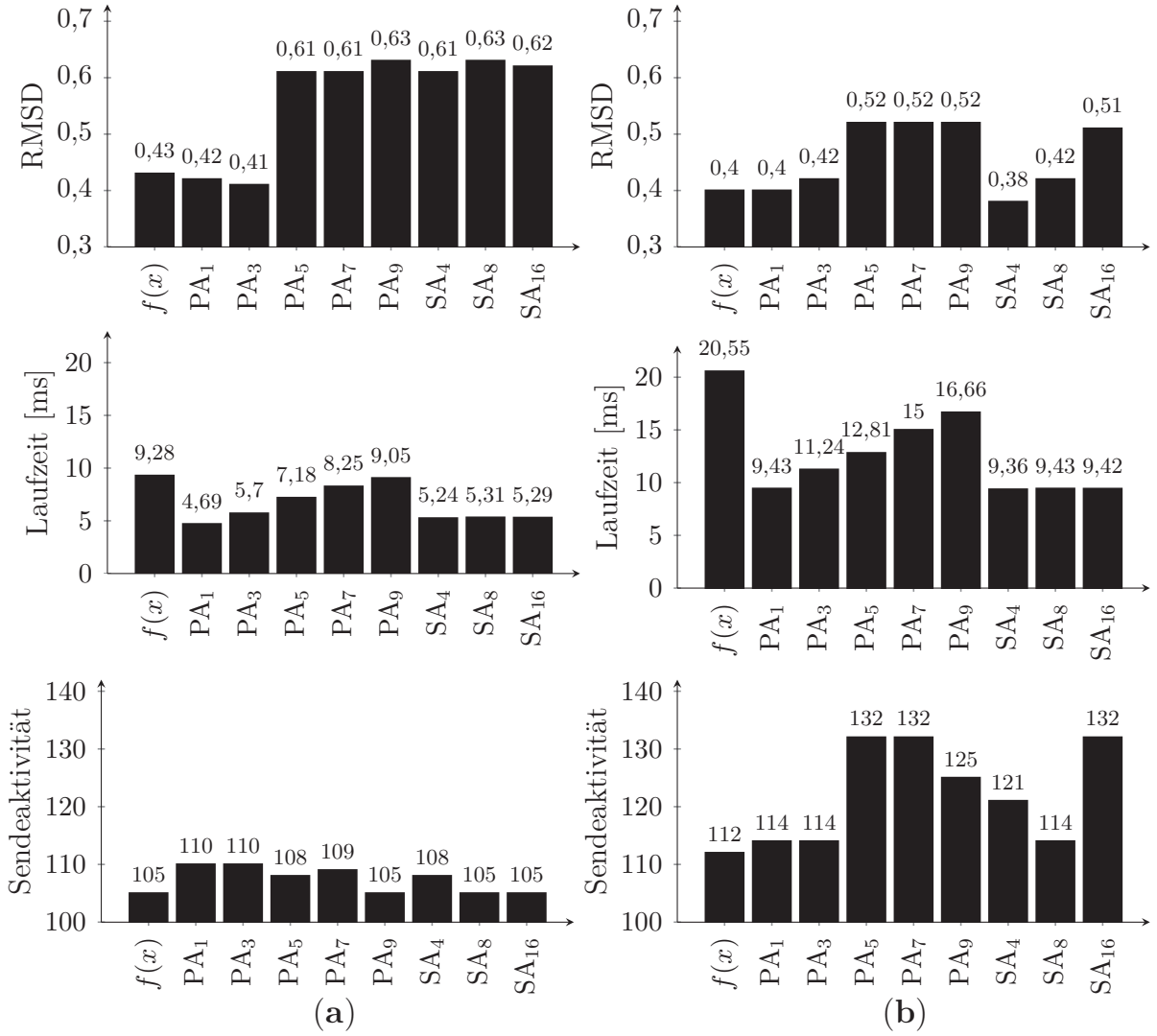


Abbildung 4.10: Evaluation der verschiedenen KNN-Prädiktoren für ein **(a)** zweischichtiges bzw. **(b)** dreischichtiges MLP mit der Sigmoidfunktion als Aktivierungsfunktion, die durch PA_n polynomiale Approximation n-ten Grades, SA_k, segmentbasierte Approximation mit k uniformen Segmenten und $f(x)$ als direkte Festkomma-Programmierung berechnet wird.

here Rechengenauigkeit der Approximationen die Sendeaktivität geringer. Bei Verwendung von drei Schichten treten hohe Unterschiede zwischen den verschiedenen Ansätzen auf, die keine Regelmäßigkeit erkennen lassen.

Prädiktionsgenauigkeit Für die Prädiktionsgenauigkeit lassen sich, ähnlich wie bei der Sendeaktivität, große Unterschiede für die verschiedenen Approximationen erkennen. Während in manchen Fällen der RMSD-Wert der Originalfunktion sogar unter-

boten wird, ist das Ergebnis, gerade für die zweischichtigen KNN-Strukturen, meistens deutlich schlechter.

Die Ergebnisse zeigen, dass durch Approximation der Sigmoidfunktion des KNN-Prädiktors die Laufzeit im Vergleich zur Originalfunktion deutlich gesenkt werden kann. Für die polynomiale Approximation steigt die Laufzeit dabei mit dem Grad des Polynoms deutlich an, bei uniformer Segmentierung ist hingegen ein vernachlässigbar geringer Anstieg zu erkennen. Hinsichtlich sowohl Sendeaktivität als auch Prädiktionsgenauigkeit erzielt jedoch keine Funktionsapproximation gute Ergebnisse. Für die uniforme Segmentierung ist allerdings eine höhere Anzahl an verwendeten Segmenten nahezu immer gleichbedeutend mit einer Verbesserung der Resultate hinsichtlich der Sendeaktivität. Jedoch treten bei Anwendung dieser Methode Sprungstellen im Funktionsverlauf der Aktivierungsfunktion auf, die zu Fehlern in der *Backpropagation* durch Aufschwingen führen können (siehe Kap. 3.7).

Zusammengefasst lässt sich erkennen, dass sich softwarebasierte KNN-Prädiktoren mit approximierter Sigmoidfunktion grundlegend zur Messung von Umweltdaten in Containern eignen. Allerdings ist weder für polynomiale Approximation noch für uniforme Segmentierung eine Konfiguration erkennbar, die in den untersuchten Fällen qualitativ die Ergebnisse der Prädiktion mit Originalfunktion heranreicht. Für die nachfolgend behandelten Hardwareumsetzungen werden daher Funktionsapproximationen mit deutlich höherer Rechengenauigkeit betrachtet.

4.4 Hardwarebasierte KNN-Prädiktion

Bei hardwarebasierter Umsetzung werden Schaltungen zur KNN-Prädiktion entworfen, die zum Einen den gegebenen Algorithmus optimiert abbilden sollen, zum Anderen die Ansteuerung der anderen Module des Sensorknotens aufrechterhalten. So kann durch Auswahl einer geeigneten Architektur und Implementierungstechnik, wie in Kap. 2.2 und Kap. 2.3 vorgestellt, eine sehr flexible Anpassung an die gewünschte Algorithmik realisiert werden. Die hierdurch zu erwartende Steigerung der Performance hinsichtlich Ausführungsgeschwindigkeit geht, wie in Kap. 4.2 erwähnt, einher mit einer deutlichen Erhöhung der Energieeffizienz. Des Weiteren soll der Fokus beim hardwarebasierten Ansatz sowohl auf Funktionsapproximationen mit hoher Rechengenauigkeit, als auch auf möglichst flexiblen Architekturen hinsichtlich der KNN-Umsetzung liegen. Im Rahmen dieser Arbeit werden hierfür CORDIC- sowie ALFA-basierte Implementierungen betrachtet (siehe Kap. 3.4.2 und Kap. 3.7). Auch für den hardwarebasierten Ansatz wird, wie in Kap. 4.3, der tabellengestützte Ansatz ausgeschlossen, da mit realistischen Datenwortbreiten ein extrem hoher Speicherbedarf einhergeht.

4.4.1 CORDIC-Hardwarebeschleuniger

Für die Hardwareumsetzung des KNN-Prädiktors durch einen CORDIC-basierten Hardwarebeschleuniger wird eine CORDIC-basierte Recheneinheit (siehe Kap. 3.4.2) zur Berechnung komplexer mathematischer Terme herangezogen [104]. Wegen der zugrunde liegenden iterativen Verarbeitungsweise soll dieser Ansatz mit möglichst hoher Rechengenauigkeit einhergehen. Der so entstehende Hardwareentwurf wird dabei in die vorliegende Signalverarbeitungseinheit als Hardwarebeschleuniger integriert, wofür eine Anpassung an den Daten- und Adressbus des MSP430 notwendig ist.

Generell kann der CORDIC-Hardwarebeschleuniger in Speicher-, Daten- und Steuerpfad unterteilt werden, die im Folgenden separat beschrieben werden. Ein Überblick über die Gesamtarchitektur ist zudem in Abb. 4.11 dargestellt.

Speicher Der CORDIC-Hardwarebeschleuniger besitzt einen Datenspeicher u.a. für die Zwischenspeicherung der Gewichtsmatrizen. Genau wie bei den softwarebasierten Ansätzen sollen für KNN-Strukturen mit maximal drei Schichten betrachtet werden. Auch ist die Datenwortbreite, gemäß den Vorgaben des MSP430 für externe Zugriffe, auf 16 Bit festgelegt. Der Datenpfad arbeitet intern im $Q2.15$ Format und benötigt somit 18 Bit, da ansonsten Überläufe für die Exponentialfunktion auftreten können. Da der daraus resultierende Speicherbedarf mit 62 Registern sehr gering ist, wird der Datenspeicher durch Register realisiert.

Datenpfad Der Datenpfad des Hardwarebeschleunigers besteht aus einer zur Laufzeit rekonfigurierbaren CORDIC-Implementierung, da sämtliche im KNN auftretenden Rechenoperatoren sich hiermit durchführen lassen (siehe Tab. 3.2). Multiplikation und Division können mittels linearer Rechenmethode berechnet werden. Bei der Umsetzung von Schiebe- und Basisoperationen kann auf die grundlegenden Rechenelemente des CORDIC zurückgegriffen werden. Die Sigmoidfunktion wird mit hyperbolischer Rechenmethode umgesetzt. Die Berechnung des Skalierungsfaktors kann, wie für die lineare Rechenmethode in Gl. (3.34), durch entsprechende Wahl der Startwerte ($x_0 = K_H$, $y_0 = 0$) entfallen.

Des Weiteren müssen die Eingangswerte an den Konvergenzbereich des CORDIC angepasst werden. Wie in Kap. 3.4.2 beschrieben liegt er für Exponentialfunktionen bei $|x| \leq 1,118$. Da bei der vorliegenden KNN-Struktur betragshöhere Werte auftreten können, wird die Sigmoidfunktion zu

$$\phi(x) = \frac{1}{1 + e^{k(-x)} \cdot e^{r(-x)}} = \frac{1}{1 + e^{-k(x)} \cdot e^{-r(x)}} \tag{4.32}$$

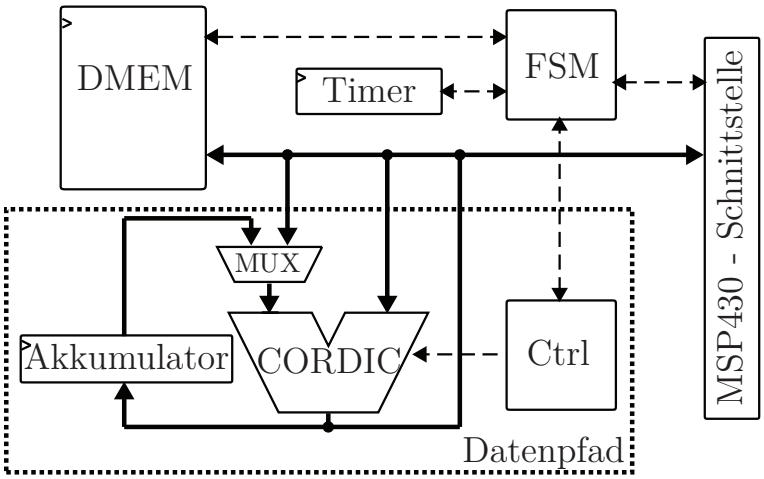


Abbildung 4.11: Hardwarearchitektur des CORDIC-Hardwarebeschleunigers zur KNN-Prädiktion. Die gestrichelten bzw. durchgezogenen Linien stellen Steuer- bzw. Datenpfade dar. Über die Schnittstelle ist der Hardwarebeschleuniger als Peripheriemodul an den Steuer- und Datenbus der MSP430-CPU angebunden.

transformiert. Der Eingangsoperand wird in einen ganzzahligen Anteil $k(x)$ und das Residuum $r(x)$ aufgeteilt, so dass

$$x = k(x) + r(x) \quad \text{mit} \quad k(x) = \lfloor x \rfloor \ , \tag{4.33}$$

mit $\lfloor \blacksquare \rfloor$ als wertbezogenem Abrunden, und

$$r(x) = \begin{cases} \mathrm{mod}_1(|x|) & , \text{wenn} \quad x \geq 0 \\ 1 - \mathrm{mod}_1(|x|) & , \text{wenn} \quad x < 0 \end{cases} \ , \tag{4.34}$$

mit $\mathrm{mod}_1(x)$ als Rest der Division durch Eins (Modulo), gilt. Um den Rechenaufwand weiter zu verringern, wird die Funktion abschnittsweise, in Abhängigkeit vom Vorzeichen, betrachtet. Durch Erweiterung ergibt sich die Rechenvorschrift der Sigmoidfunktion zu

$$\phi(x) = \begin{cases} \frac{e^{r(x)}}{e^{r(x)}+e^{-k(x)}} & , \text{wenn} \quad x \geq 0 \\ \frac{e^{k(x)}}{e^{k(x)}+e^{-r(x)}} & , \text{wenn} \quad x < 0 \end{cases} \ , \tag{4.35}$$

wodurch die Multiplikation im Nenner entfällt. Durch diese Transformation vereinfacht sich der tatsächliche Rechenaufwand der Exponentialfunktion zu zwei CORDIC-Operationen und einer abschließenden Addition (siehe Tab. 4.3).

Tabelle 4.3: Ablauf der FSM für *Forward Prediction* mit **(a)** der Gewichtung der Eingangsdaten x_i und der Berechnung der Sigmoidfunktion für **(b)** positive bzw. **(c)** negative Operanden des j-ten Neurons in der m-ten Schicht. FSM gibt den Wert der FSM im Binärsystem an, aus dessen Bits sich die Speicheradressen beider Datenquellen (SRC 1, SRC 2) bzw. der -senke (DST) ableiten lassen.

Operation	FSM	Inkrement	SRC 1	SRC 2	DST
Multiplikation	$m_1\ m_0 j_1 j_0 0\ 0 i_1 i_0 0$	1	$w_{m,ij}$	$x_{m,ij}$	`ACCU`
Addition	$m_1\ m_0 j_1 j_0 0\ 0 i_1 i_0 1$	1	`ACCU`	$u_{m,j}$	$u_{m,j}$

(a)

Operation	FSM	Inkrement	SRC 1	SRC 2	DST
Exponentialfunktion	$m_1\ m_0 j_1 j_0 0\ 1000$	8	$u_{m,j}$	-	`ACCU`
Addition	$m_1\ m_0 j_1 j_0 1\ 0000$	8	$-k(x)$	`ACCU`	$x_{m+1,j}$
Division	$m_1\ m_0 j_1 j_0 1\ 1000$	8	`ACCU`	$x_{m+1,j}$	$x_{m+1,j}$

(b)

Operation	FSM	Inkrement	SRC 1	SRC 2	DST
Exponentialfunktion	$m_1\ m_0 j_1 j_0 0\ 1000$	8	$u_{m,j}$	-	`ACCU`
Addition	$m_1\ m_0 j_1 j_0 1\ 0000$	8	$-k(x)$	`ACCU`	`ACCU`
Division	$m_1\ m_0 j_1 j_0 1\ 1000$	8	$-k(x)$	`ACCU`	$x_{m+1,j}$

(c)

Wie in Gl. (4.33) angegeben wird $k(x)$ durch Abrunden bestimmt, weswegen $r(x)$ für negative Zahlen invertiert werden muss. Um den Rechenaufwand auch hierbei zu minimieren, ist diese Operation durch Invertierung des Vorzeichenbits und Bildung des Einerkomplements realisiert, was durch einfache Verwendung von Invertern umgesetzt werden kann. $k(x)$ wird durch fest verdrahtete Konstanten und einen Multiplexer realisiert. Der Zahlenbereich ist dabei auf $e^{-4} \leq e^{-k(x)} \leq e^{0}$ begrenzt. Für das Residuum ergibt sich $e^{0} \leq e^{r(x)} \leq e^{1} - 2^{-15}$. Die vollständige CORDIC-basierte Berechnung der Sigmoidfunktion ist somit mittels einer hyperbolischen (Exponentialfunktion) und einer linearen CORDIC-Operation sowie einer Addition durch

$$\phi(x) = \begin{cases} \frac{e^{r(x)}}{e^{r(x)}+e^{-k(x)}} & \text{, wenn } \ x \geq 0 \\ \frac{e^{k(x)}}{e^{k(x)}+e^{r(|x|_{1,\overline{v}})}} & \text{, wenn } \ x < 0 \end{cases} \quad , \tag{4.36}$$

mit $|\,\blacksquare\,|_{1,\overline{v}}$ als Einerkomplement mit invertiertem Vorzeichenbit, möglich.

Zur Konfiguration des Datenpfades für jede einzelne Operation wird eine interne CORDIC-Ansteuereinheit verwendet, die sowohl für den zeitlichen Ablauf als auch die dazugehörige Verschaltung elementarer Komponenten zuständig ist. Zudem besitzt der Datenpfad ein Akkumulator-Register `ACCU`, das für das Ablegen von Zwischenergebnissen verwendet werden kann. Darüber hinaus werden *Clock-Gating* und *Operand-Isolation* als zusätzliche Implementierungstechniken zur Senkung des Energieverbrauchs im Hardwareentwurf berücksichtigt und automatisch bei der Logiksynthese eingefügt.

Steuerpfad Der Steuerpfad besteht aus einer FSM, die für die Verschaltung des Datenpfades zur Laufzeit zuständig ist, sowie einen Konfigurationsmodul zur Anbindung des Hardwarebeschleunigers an den übergeordneten Mikrocontroller. Letzteres ermöglicht den Zugriff auf die Speicherressourcen sowie die grundlegende Einstellung des Ablaufs. Die FSM ist als registerbasierter Zähler implementiert, dessen Wert den aktuellen Zustand bestimmt. Durch Variation des Inkrements können unterschiedliche Abfolgen an Operationen realisiert werden. Die FSM lässt sich in zwei Hierarchieebenen unterteilen, die die globale und lokale Ansteuerung des Datenpfades regeln. Beiden Ebenen können somit verschiedene Bitsequenzen des Zählers zugewiesen werden. Die globale Ebene kann in vier Abschnitte unterteilt werden: *Forward Prediction* und *Backpropagation* sowie Vor- und Nachbereitung der Daten. Sowohl die notwendigen Steuersignale zur Konfiguration des Datenpfades als auch die Auswahl der Quell- und Zieloperanden sind direkt aus dem aktuellen Zustand des Zählers extrahierbar. Der beispielhafte Ablauf der *Forward Prediction* ist in Tab. 4.3 gegeben.

4.4.2 ALFA-ASIP

Neben der Implementierung eines Hardwarebeschleunigers soll außerdem die Umsetzung des KNN-Prädiktors als ASIP betrachtet werden [102]. Ausgehend von den algorithmischen Anforderungen muss demnach ein angepasster Instruktionssatz entwickelt werden, der einen entsprechend optimierten Datenpfad ansteuert. Da diese Architektur einen äußerst flexiblen Ansatz darstellt, soll die spätere Programmierung des ASIP unterschiedliche Varianten des KNN-Algorithmus unterstützen, sowohl hinsichtlich der Neuronen als auch der Topologie. Des Weiteren muss auch die Ansteuerung der übrigen Module des Sensorknotens gewährleistet sein, d.h. der ASIP muss über entsprechende Ein- und Ausgänge sowie Signalverarbeitungseinheiten verfügen. Die Taktrate des ASIP wird entsprechend der Vorgabe des im TelosB verbauten MSP430 auf 4 MHz gesetzt. Diese im Verhältnis geringe Taktfrequenz erlaubt eine Instruktionsabarbeitung in i.d.R. einem Takt und ohne den Einsatz von *Pipeline*-Stufen (siehe Kap. 2.3). Für die Berechnung der Multiplikation und Division werden jeweils zwei Taktzyklen benötigt.

Zudem ist es möglich, Konstanten direkt im Instruktionswort anzugeben, was in 32 Bit breiten Instruktionen resultiert. Auch hierfür sind zwei Takte zur Abarbeitung erforderlich. Ein vollständiger Überblick über den Instruktionssatz findet sich in Anhang A.5.1.

Für die Berechnung aufwändiger algebraischer Terme wird auf die in Kap. 3.7 eingeführte ALFA-basierte nicht-uniforme Funktionsapproxmiation zurückgegriffen. Eine detaillierte Beschreibung des zugehörigen Daten- und Steuerpfades sowie der Speichermodule und Peripherie ist im Folgenden gegeben (siehe auch Abb. 4.12).

Speicher Der KNN-ASIP besitzt zwei unterschiedliche Speichermodule, einen 2 kByte großen Programmspeicher und einen 128 Byte umfassenden Datenspeicher, die als SRAM bzw. durch Register realisiert sind. Auch für die ASIP-Implementierung ist die Datenwortbreite auf 16 Bit festgelegt. Der Programmspeicher ist nur während einer speziellen Initialisierungsphase, die vor der eigentlichen Befehlsverarbeitung abläuft, beschreibbar. Auf dem Datenspeicher kann ausschließlich indirekt durch Verwendung von Zeigern (engl.: *pointer*) zugegriffen werden. Nähere Informationen zu dieser Art der Datenverarbeitung finden sich im Abschnitt Datenpfad.

Neben den globalen Speicherressourcen besitzt der KNN-ASIP zudem 16 CPU-interne Register, die allgemein als schnelle Zwischenspeicher verwendet werden können, aber auch spezielle Funktionalität aufweisen. Neben üblichen Spezialregistern, wie z.B. dem Programmzeiger (engl.: *program counter*, PC) und Konfigurationsregistern, werden auch die vorangehend angesprochenen Zeiger durch die CPU-Register realisiert. Eine Beschreibung der zugehörigen Ansteuerung findet sich im Abschnitt Steuerpfad. Sämtliche Register und ihre Funktionen sind in Anhang A.5.2 aufgelistet.

Datenpfad Der Datenpfad ist für die eigentliche Ausführung der arithmetischen und logischen Instruktionen zuständig. Neben Addition und Subtraktion sind zudem logisches UND, ODER, XOR, NOT und der Schiebeoperator berücksichtigt. Für die anwendungsspezifische Anpassung sind weiterhin Multiplikation, Division, Sigmoidfunktion und hyperbolischer Tangens vorhanden, die, wie vorangehend angesprochen, mittels automatischer Funktionsapproximation erstellt werden. Dabei wird für die Aktivierungsfunkionen die in Kap. 4.3 erwähnte Skalierung des Eingangsoperanden vorgenommen. Es gilt somit die modifizierte Sigmoidfunktion aus Gl. (4.22). Analog ergibt sich für den hyperbolischen Tangens

$$\tilde{\phi}(x) = \tanh(4x) \ . \tag{4.37}$$

Aufgrund der Punktsymmetrie wird zudem nur der Funktionsverlauf $x \geq 0$ approximiert, da durch Invertierung der MSB bzw. Bildung des Einerkomplements $\tilde{\phi}(x)$ näherungsweise gespiegelt werden kann.

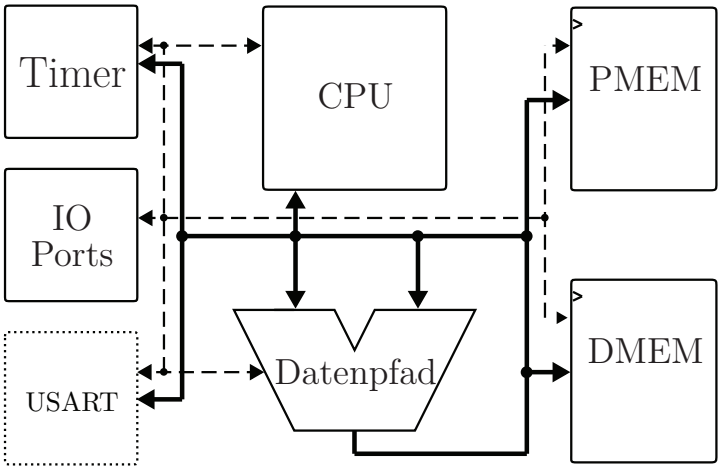


Abbildung 4.12: Architektur des ALFA-ASIP für KNN-Prädiktion. Die gestrichelten bzw. durchgezogenen Linien kennzeichnen den Steuer- bzw. Datenpfad. Das gepunktete USART-Modul ist u.a. für die Anbindung der Sensorik an den Sensorknoten notwendig.

Um die Multiplikation und Division durch Funktionsapproximationen zu berechnen, ist ein Wechsel des Zahlenformats notwendig. So werden die Operanden zunächst ins logarithmische Zahlenformat transformiert (siehe Kap. 3.1.3), wodurch sich der Rechenaufwand auf eine Addition bzw. Subtraktion reduziert. Durch anschließende Rücktransformation ergibt sich ein Ergebnis im gewünschten Festkomma-Zahlenformat. Dieses Vorgehen lässt sich mathematisch durch

$$x_1 \cdot x_2 = N^{log_N(x_1)+log_N(x_2)} \tag{4.38}$$

und

$$\frac{x_1}{x_2} = N^{log_N(x_1)-log_N(x_2)} \tag{4.39}$$

ausdrücken. Die Funktionsapproximation wird somit nicht direkt zur Umsetzung der eigentlichen Operation, sondern für die Umrechnungsfunktionen der Zahlendarstellung eingesetzt. Eine vor- bzw. nachgestellte Skalierung des Festkommawerts erlaubt die Verwendung der in [79] für $N = 2$ formulierten Gleichungen

$$m_L = \mathrm{ld}(x_F + 1) \tag{4.40}$$

für die Konvertierung ins logarithmische Zahlenformat (LOG(x)) bzw.

$$x_F = 2^{m_L} - 1 \tag{4.41}$$

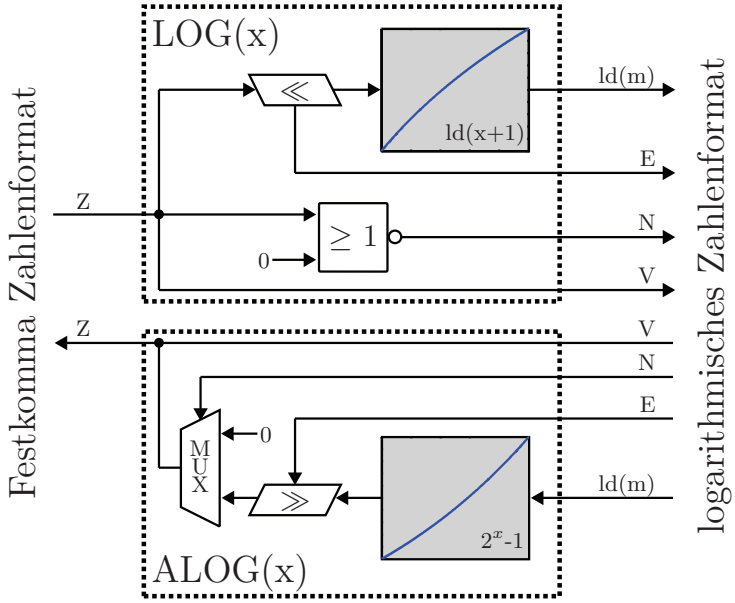


Abbildung 4.13: Hardwareumsetzung der in Gl. (4.40) bzw. Gl. (4.41) beschriebenen LOG(x)- bzw. ALOG(x)-Konverter. Weitere verwendete Kürzel beziehen sich auf die in Kap. 3.1.3 gegebene mathematische Definition. Die Funktionen $\mathrm{ld}(x-1)$ und 2^x-1 werden mittels der in Kap. 3.7 vorgestellten automatischen Funktionsapproximation realisiert.

für die Konvertierung ins Festkomma-Zahlenformat (ALOG(x)). m_L bzw. x_F entsprechen dabei der logarithmischen Mantisse bzw. dem Festkommawert. Der Einsatz dieser Umrechnungsfunktionen bringt mehrere Vorteile mit sich, wie z.B. Werte- und Funktionsbereiche zwischen Null und Eins oder gute Anwendbarkeit der Funktionsapproximation [24]. Die Skalierung erfolgt durch Schiebeoperationen, deren Anzahl darüber hinaus identisch ist mit dem Exponenten im logarithmischen Zahlenformat. Die Steuerbits für negative Werte und Null können direkt und durch Komparatoren ermittelt werden. Der Hardwareaufbau für den LOG(x)- bzw. ALOG(x)-Konverter ist in Abb. 4.13 dargelegt.

Steuerpfad Hauptbestandteil des Steuerpfades ist die CPU, deren Aufgabe das Dekodieren der Instruktionen sowie das Erstellen von Ansteuersignalen ist. Die Instruktionen sind dabei in Daten- und Steuerbefehle unterteilbar, wobei bei Ersteren zudem zwischen Ein- und Zweioperand-Befehlen unterschieden werden kann, die für die Umsetzung der entsprechenden Operationen im Datenpfad zuständig sind. Die Steuerbefehle regeln den Programmablauf, d.h. sie ermöglichen sowohl bedingte als auch unbedingte Sprünge. Bei Einoperand-Befehlen fungiert ein Operand sowohl als Datenquelle als auch als -senke. Bei Zweioperand-Befehlen tritt ein *Immediate*-Operand als Datenquelle auf.

Für die effiziente Verarbeitung von KNN-Algorithmen besitzt der ASIP vier verschiedene Arten der Operand-Adressierung, die u.a. die Berechnung von Vektor- oder Matrixmultiplikationen beschleunigen. Die erste ist die direkte Adressierung, bei der nur auf die 16 CPU-Register zugegriffen werden kann. Die Datenquelle oder -senke wird dabei in der Instruktion festgelegt. Für die zweite, die indirekte Adressierung wird das angegebene CPU-Register als Zeiger auf den Speicher verwendet. Diese Zugriffsart ist durch das Operand-Prefix @ gekennzeichnet. Die dritte Form der Adressierung behandelt die vorangehend erwähnten *Immediate*-Operanden: Die Datenquelle ist hierbei direkt in der Instruktion angegeben, wodurch beliebige Konstanten in den Speicher oder in die CPU-Register geschrieben werden können. Allerdings werden zwei Takte zum Einlesen der zugehörigen Instruktion benötigt. Die häufig auftretenden Konstanten Null, Eins, Zwei und Vier können auch durch speziellen Zugriff auf CPU-Register erhalten werden, wodurch die Verarbeitungszeit nur einen Takt beträgt (siehe Anhang A.5.3). Die vierte Art von Adressierung legt die Datensenke auf das `ACCU`-Register fest (Postfix *_acc*), wodurch mit drei Operanden gearbeitet werden kann. Für den ersten

Algorithmus 4.2: Beispielhafter Assemblercode zur Neuron-Berechnung

```
; Initialisierung
mov addr_w3, AIR0          ; Pointer auf Adresse von w3 setzen
mov addr_x3, AIR1          ; Pointer auf Adresse von x3 setzen
mov addr_y3, AIR2          ; Pointer auf Adresse von y3 setzen
mov -1, +AIR0-             ; Automatische Verringerung des Pointers um 1
mov -1, +AIR1-             ; Automatische Verringerung des Pointers um 1
; Laufzeit
mul_acc @AIR0, @AIR1       ;  ACCU = x3 · w3
mov ACCU, R5               ;  R5 = R5 + ACCU
shr R5, 1                  ;  R5 = R5 /2
mul_acc @AIR0, @AIR1       ;  ACCU = x2 · w2
shr ACCU, 1                ;  ACCU = ACCU /2
add ACCU, R5               ;  R5 = R5 + ACCU
shr R5, 1                  ;  R5 = R5 /2
mul_acc @AIR0, @AIR1       ;  ACCU = x1 · w1
shr ACCU, 2                ;  ACCU = ACCU /4
add ACCU, R5               ;  R5 = R5 + ACCU
mul_acc @AIR0, @AIR1       ;  ACCU = x0 · w0
shr ACCU, 2                ;  ACCU = ACCU /4
add ACCU, R5               ;  R5 = R5 + ACCU
smf R5, @AIR2              ;  Sigmoidfunktion von R5
```

Operanden ist in diesem Modus indirekte Adressierung automatisch aktiv. Der zweite Operand hingegen erlaubt sowohl direkten als auch indirekten Zugriff.

Neben der beschriebenen indirekten Adressierung ist es zudem möglich, den Zeigerwert automatisch vor bzw. nach einer ausgeführten Operation zu erhöhen bzw. zu verringern. So besitzen vier spezielle CPU-Register (+`AIR0`− - +`AIR3`−) die Eigenschaft, den Inhalt vier zugeordneter Register (`AIR0 - AIR3`) als vorstehendes Inkrement oder nachstehendes Dekrement ihres Zeigerwerts zu behandeln. Durch diese Funktion ist es möglich, die häufig auftretenden Vektor- oder Matrixmultiplikationen sehr effizient zur Laufzeit abzuarbeiten. Ein Beispiel hierfür ist in Alg. 4.2 gegeben. Ein Überblick über die Adressierungsmodi findet sich in Anhang A.5.3.

4.4.3 Evaluation

Die hardwarebasierten KNN-Prädiktoren werden sowohl hinsichtlich ihrer Rechen- als auch ihrer Prädiktionsgenauigkeit, Laufzeit und Implementierung untersucht und bewertet, wofür wieder auf die in Abb. 4.7 gegebene Messkurve zurückgegriffen wird. Dabei sollen ausgehend von den Ergebnissen der softwarebasierten Evaluation (siehe Kap. 4.3.3) möglichst hohe Rechengenauigkeiten und daraus folgend hohe Prädiktionsgenauigkeiten erzielt werden. Zur Auswertung dieser beiden Aspekte wird auf ein festkommabasiertes Matlab-Modell zurückgegriffen, das auch als Spezifikations- und Verifikationsreferenz für den Hardwareentwurf fungiert. Latenz und Performance werden anhand einer entsprechenden HDL-Beschreibung ermittelt. Des Weiteren werden für den CORDIC-Hardwarebeschleuniger Datenpfade mit einem *Loop-Unrolling* Faktor von Eins (CORDIC_{LU1}) bzw. Zwei (CORDIC_{LU2}), also mit einem bzw. zwei Iterationsschritten pro Takt, betrachtet.

Rechengenauigkeit Um die aus den Ergebnissen der softwarebasierten Untersuchungen geforderte hohe Rechengenauigkeit zu erreichen, sollen beim CORDIC-Hardwarebeschleuniger 16 Iterationsschritte für die CORDIC-Operationen Exponentialfunktion, Division und Multiplikation verwendet werden. Durch die doppelten Iterationsschritte der hyperbolischen Rechenmethode (siehe Kap. 3.4.2) und die abschließende Addition ergeben sich für die Exponentialfunktion insgesamt 18 benötigte Rechenschritte (siehe Tab. 3.2b). Für den ALFA-ASIP werden Approximationen mit kontinuierlichem Funktionsverlauf verwendet, um mögliches Fehlverhalten z.B. durch Aufschwingen zu vermeiden. Als Randbedingungen der ALFA-Synthese werden des Weiteren eine mittlere Rechengenauigkeit von $\text{mae}_{ALFA} = 2^{-13}$ sowie ein Quantisierungsfaktor von Zwei gewählt, was sich durch experimentelle Analyse als ausgewogene Vorgabe hinsichtlich Signalverarbeitungsaufwand und Komplexität erweist. Da sowohl der hyperbolische Tangens als auch die Sigmoidfunktion punktsymmetrisch sind, wird in beiden

Tabelle 4.4: Bewertung der Rechengenauigkeit der in **(a)** Kap. 4.4.1 und **(b)** Kap. 4.4.2 benötigten elementaren Funktionen anhand des mae-Werts (siehe Kap. 3.7). Die ALFA-basierte Funktionsapproximation weist dabei keine Sprungstellen auf, d.h. der Funktionsverlauf ist kontinuierlich.

Funktion	QF	Segmente	mae_{ALFA}
Sigmoidfunktion	2	147	$4,39 \cdot 10^{-5}$
Hyperbolischer Tangens	2	299	$4,06 \cdot 10^{-5}$
$\text{ld}(x+1)$	2	381	$4,92 \cdot 10^{-5}$
$2^x - 1$	2	376	$5,10 \cdot 10^{-5}$

(a)

Funktion	Latenz	mae_{CORDIC}
Exponentialfunktion	18/9	$1,53 \cdot 10^{-5}$
Multiplikation	16/8	$7,06 \cdot 10^{-6}$
Division	16/8	$7,07 \cdot 10^{-6}$

(b)

Fällen nur der positive Funktionsabschnitt approximiert. Negative Eingangswerte können durch Bildung des Einerkomplements näherungsweise gebildet werden.

Beim Vergleich der resultierenden Rechengenauigkeit beider Ansätze weist der CORDIC-Hardwarebeschleuniger bessere Ergebnisse als der ALFA-ASIP auf. Ein Überblick über die Ergebnisse ist in Tab. 4.4 gegeben.

Prädiktionsgenauigkeit Als zweiter Schritt wird die Prädiktionsgenauigkeit für dynamische Messintervalle für beide Ansätze, unter Verwendung der in Abb. 4.8 gegebenen Messkurve, bestimmt. Beide Ansätze erreichen im Vergleich zur Festkomma-Programmierung der Originalfunktion dieselbe Anzahl an Messpunkten (105 für den zwei- bzw. 112 für den dreischichtigen Ansatz) sowie nur vernachlässigbar schlechtere RMSD-Werte.

Laufzeit Zur Auswertung der Laufzeit des KNN-Algorithmus wird die Latenz, also die Anzahl der benötigten Takte zur vollständigen Abarbeitung des Algorithmus, als Bewertungsgrundlage herangezogen. Dabei wird ausschließlich der KNN-Prädiktor mit dreischichtigem MLP betrachtet. Nach Tab. 4.4b benötigt der CORDIC-basierte Ansatz 18 bzw. 9 oder 16 bzw. 8 Takte für die Berechnung der Division, Multiplika-

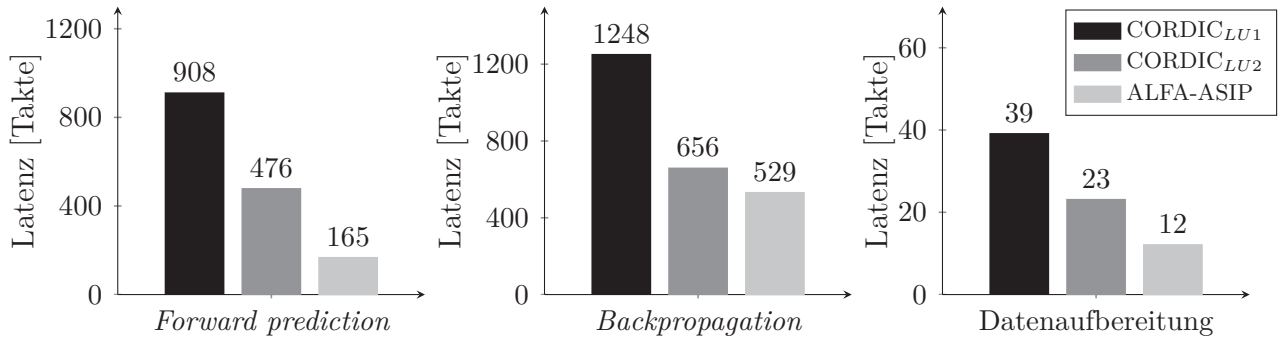


Abbildung 4.14: Evaluation der hardwarebasierten KNN-Prädiktoren hinsichtlich Laufzeit, gemessen an der Anzahl an Takten für *Forward Prediction*, *Backpropagation* und Datenaufbereitung.

tion oder Exponentialfunktion für CORDIC$_{LU1}$ bzw. CORDIC$_{LU2}$. Der ALFA-ASIP wird durch eine entsprechende Assembler-Umsetzung des KNN-Prädiktors programmiert (siehe Alg. 4.2). Die Funktionsapproximationen werden in jeweils einem Takt berechnet. Multiplikation und Division lassen sich, wie in Kap. 4.4.2 erläutert, durch das logarithmische Zahlenformat berechnen. Um die Latenz möglichst gering zu halten, werden zwei LOG(x) Konverter, also einer für jeden Eingangsoperanden, verwendet. Für die Rücktransformation des Ergebnisses wird hingegen nur ein ALOG(x) Modul benötigt. Die gesamte Rechenoperation besitzt somit eine Sequenz von fünf Addierern (auch algorithmische Latenz genannt), die auf insgesamt zwei Takte aufgeteilt werden können. Somit benötigt ALFA-ASIP auch zur Berechnung der Multiplikation und Division insgesamt zwei Takte. Neben der Signalverarbeitung im Datenpfad besitzt der ALFA-ASIP zusätzlich Steuerbefehle, z.B. für Programmschleifen, die einen oder zwei Takte benötigen (siehe Kap. 4.4.2).

Die Ergebnisse der Laufzeitevaluation weisen den ALFA-ASIP im Vergleich als schnellsten hardwarebasierten Ansatz aus. Abb. 4.14 gibt einen detaillierten graphischen Überblick.

Implementierung Die beschriebenen KNN-Prädiktoren werden gemäß dem in Kap. 2.4.2 gegebenen Ablauf in ein Layout synthetisiert. Für den CORDIC-Hardwarebeschleuniger wird eine VHDL-Implementierung erstellt, die in eine bestehende HDL-Beschreibung einer MSP430 CPU integriert wird. Der KNN-ASIP wird durch den *Processor Designer* der Firma Synopsys implementiert. Der ALFA-basierte Datenpfad wird mittels ALFA-Synthese erstellt und in den Entwurf integriert. Das in Abb. 4.12 gezeigte USART-Modul ist nur für den tatsächlichen Einsatz am Sensorknoten notwendig und wird daher an dieser Stelle nicht berücksichtigt. Beide Ansätze werden mittels Logik- und Layoutsynthese in physikalische Beschreibungen übersetzt und anhand der angesprochenen Matlab-Modelle validiert. Als Fertigungstechnologie wird

Tabelle 4.5: Performance-Ergebnisse der 130nm UMC Faraday Layoutsynthese der KNN-Prädiktoren als CORDIC-Hardwarebeschleuniger und als ALFA-ASIP.

Referenz	Taktfrequenz [MHz]	Komplexität [kGE]	Leistungsverbrauch [μW]	Energieeffizienz [nJ/KNN]
CORDIC$_{LU1}$	4	11,88	92,49	50,75
CORDIC$_{LU2}$	4	12,92	109,60	31,65
ALFA-ASIP	4	58,67	221,00	39,00

auf den 130nm Faraday Prozess der Firma UMC zurückgegriffen [119]. Zur Auswertung werden die Schaltungskomplexität, der Leistungsverbrauch sowie der Energieverbrauch pro KNN-Berechnung betrachtet, die in Tab. 4.5 zusammengefasst dargestellt sind.

Die Ergebnisse weisen zunächst den ALFA-ASIP als energieeffizienten Ansatz zur hardwarebasierten Umsetzung des KNN-Prädiktors auf, obgleich die CORDIC-basierten Ansätze im direkten Vergleich der vorgestellten Hardwareentwürfe bessere Resultate aufweisen, was durch die gewählte Hardwarearchitektur zu erklären ist. So müssen für den Einsatz auf dem Sensorknoten die CORDIC-Hardwarebeschleuniger als Peripherie in den MSP430 integriert werden, was die Performance deutlich senken würde. Der ALFA-ASIP hingegen weist eine flexible Hardwarearchitektur auf und kann theoretisch als vollwertige Signalverarbeitungseinheit eines Sensorknotens verwendet werden.

4.5 Zusammenfassung

In diesem Kapitel wurden software- und hardwarebasierte Ansätze zur effizienten Berechnung von KNN-Algorithmen vorgestellt und anhand eines vorgegebenen Einsatzgebietes, der sensorbasierten Prädiktion von Messdaten im Container, bewertet. Da eine herkömmliche Berechnung des KNN-Algorithmus aufgrund der geringen Rechenleistung von Sensorknoten unzureichende Resultate hinsichtlich Laufzeit erreicht, was sich wiederum negativ auf die Energieeffizienz auswirkt, wurde auf verschiedene Ansätze der in Kap. 3 vorgestellten Methoden zur effizienten Berechnung elementarer Funktionen zurückgegriffen. Die softwarebasierte Umsetzung polynom- und segmentbasierter Ansätze erzielt zwar eine deutliche Verbesserung der Latenz, kann jedoch keine durchgehend zufriedenstellenden Ergebnisse hinsichtlich der Prädiktionsgenauigkeit liefern. Bei der hardwarebasierten Umsetzung kommen daher Ansätze mit hoher Rechengenauigkeit zum Einsatz, was zum Einen durch eine hohe Anzahl an Iterationsschritten (CORDIC), zum Anderen durch eine hohe Anzahl an Segmenten und daraus resul-

tierend eine hohe Approximationsgenauigkeit (ALFA) erreicht wurde. Für die beiden untersuchten Ansätze wurden sehr gute Ergebnisse hinsichtlich Prädiktionsgenauigkeit erzielt. Zudem konnte die Latenz deutlich gegenüber den softwarebasierten Ansätzen gesenkt werden, wobei der ALFA-ASIP sich insgesamt als ausgewogener Ansatz hinsichtlich Energieeffizienz, Flexibilität und Rechengenauigkeit präsentiert. Die in Kap. 3.7 vorgestellte automatische Funktionsapproximation stellt somit einen effizienten Ansatz zur Signalverarbeitung mit hoher Rechengenauigkeit dar.

5 Signalverarbeitung elementarer Funktionen im Mobilfunk

Die Entwicklung optimierter Sende- und Empfangsstrukturen im Mobilfunk ist aufgrund der stetig steigenden Anforderungen ein wichtiges aktuelles Forschungsgebiet [95]. So stellt der nach wie vor ungebrochene Anstieg des Datenaufkommens, z.B. durch die immer größere Zahl an Mobilfunkteilnehmern, eine besondere Herausforderung für aktuelle und zukünftige Mobilfunksysteme dar [1]. Des Weiteren ist, genau wie in Kap. 4, die Minimierung des Energieverbrauchs ein wichtiger Aspekt für batteriebetriebene Mobilfunkendgeräte, z.B. Smartphones. So bedeutet die Steigerung der Energieeffizienz i.d.R. eine entsprechend höhere Akkulaufzeit. Einen Ansatz zur Erhöhung der Datenrate stellen u.a. Mehrantennensysteme wie MIMO (engl.:*multiple input multiple output*) dar, deren Einsatz jedoch mit einer deutlichen Erhöhung des Signalverarbeitungsaufwands einhergeht [11]. Zudem wird in aktuellen Übertragungssystemen das Mehrträgerverfahren OFDM (engl.:*orthogonal frequency division multiplex*) verwendet, durch das ein frequenzselektiver Kanal in mehrere nicht-frequenzselektive Kanäle unterteilt [129].

Die performante Implementierung ausgewählter Signalverarbeitungsschritte wird, unter Berücksichtigung der in Kap. 2.4.1 angesprochenen Verfahren zur näherungsweisen Berechnung elementarer Funktionen, in Kap. 5.2 und Kap. 5.3 detailliert vorgestellt. Zuvor ist eine Übersicht über grundlegende Aspekte der Nachrichtenübertragung gegeben.

5.1 Nachrichtenübertragung

Unter dem Begriff Nachrichtenübertragung ist allgemein die Übertragung von Nutzdaten zwischen einer Sende- und einer Empfangseinheit zu verstehen, was sich im Rahmen dieses Kapitels nur auf drahtlose Systeme beschränken soll [56]. Das grundlegende Prinzip des Senders sieht dabei zunächst die Modulation eines Signals auf eine Trägerwelle, also eine spektrale Verschiebung in ein vorgesehenes Frequenzband, vor. Auf Empfängerseite wird das reelle Bandpasssignal in ein komplexwertiges Tiefpass-

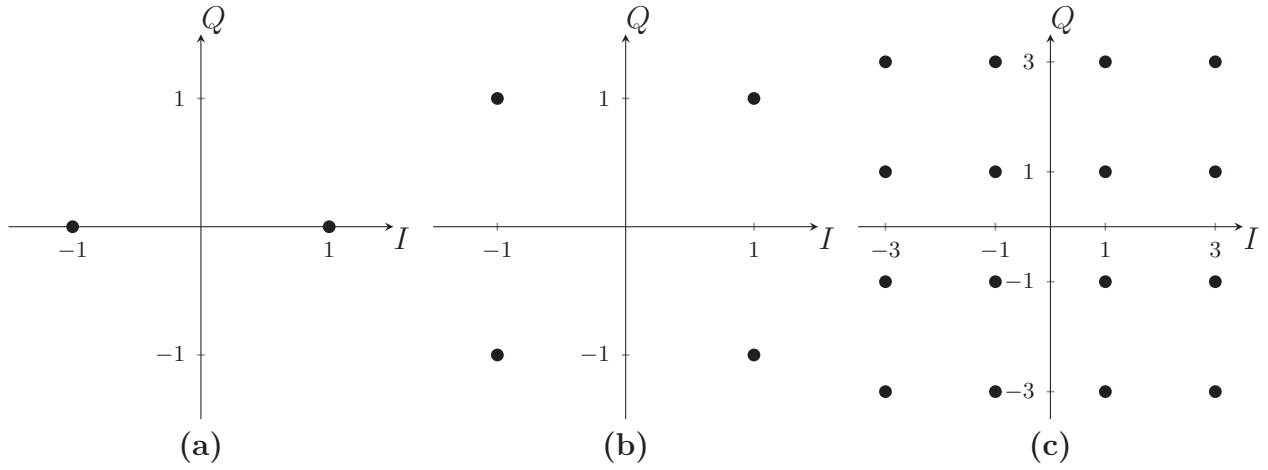


Abbildung 5.1: Signalraumkonstellationen einer **(a)** binären PSK (engl.: *phase shift keying*), **(b)** 4-QAM und **(c)** 16-QAM. I bzw. Q bezeichnen dabei den Real- bzw. Imaginärteil der Sendesymbols.

oder auch Basisbandsignal zurücktransformiert. Moderne digitale Nachrichtenübertragungssysteme verwenden zudem i.d.R. digitale Modulationsverfahren oder Mehrfachzugriffsverfahren. Beide grundlegenden Ansätze werden daher im Folgenden vorgestellt, bevor der prinzipielle Aufbau zunächst eines MIMO- und darauf folgend eines OFDM-Übertragungssystems erläutert wird.

5.1.1 Digitale Modulation

Die digitale Modulation dient der Abbildung einer diskreten Datenfolge, also z.B. Bitsequenzen der Länge m, auf sogenannte Symbole [92]. Hierfür werden besagte Bitsequenzen zunächst in Bitgruppen zusammengefasst, die jeweils feste Signalraumzuordnungen besitzen. Die so entstehenden Symbole sind zumeist komplex und setzen sich aus einer Normal- und einer Quadraturkomponente, den sogenannten I- bzw. Q-Daten (engl.: *in-phase* und *quadrature*), zusammen. Die Menge an Symbolen M ist dabei abhängig von der Länge der Bitsequenzen ($M = 2^m$) [56]. Mit einer höheren Anzahl verwendeter Symbole steigt einerseits die Effizienz der Datenübertragung und folglich auch der Datendurchsatz, andererseits ist auch die Störanfälligkeit, z.B. durch Rauscheffekte, größer.

Allgemein existiert ein Vielzahl verschiedener linearer und nicht-linearer Modulationsarten, die sich auf die grundlegenden Ansätze Amplituden-, Phasen- und Frequenzmodulation zurückführen lassen. Durch Kombination der Amplituden- und Phasenmodulation ergibt sich die M-stufige Quadratur Amplituden Modulation (engl: *quadrature amplitude modulation*, QAM), die eine heutzutage gängige Modulationsart darstellt und daher in einer Vielzahl moderner Übertragungsstandards, z.B. LTE oder WLAN,

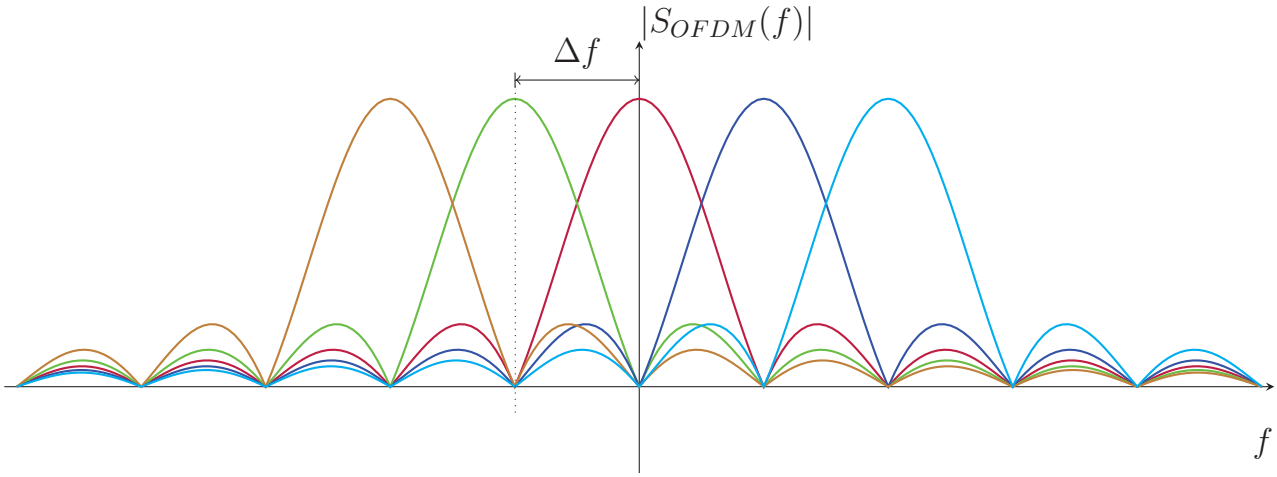


Abbildung 5.2: Prinzipielle Struktur überlappender Betragsspektren eines OFDM-Signals (Frequenzversatz Δf). Durch die orthogonale Anordnung fallen die Maxima auf die Nulldurchgänge anderer Subträger.

verwendet wird [16]. Ein Überblick über verschiedene Signalraumkonstellationen ist in Abb. 5.1 gezeigt.

5.1.2 Multiplexverfahren

Um eine möglichst hohe Auslastung des Übertragungskanals zu erreichen, ist die effiziente Nutzung der zur Verfügung stehenden Bandbreite im Mobilfunk von essentieller Bedeutung [125]. Einen gängigen Ansatz hierfür stellt die Verwendung von Multiplexverfahren dar. Grundlegende Idee ist dabei, die zur Verfügung stehenden Ressourcen, z.B. durch geeignete Aufteilung des Frequenzbands, besser auszulasten.

Darüber hinaus ermöglichen Multiplexverfahren auch den Mehrteilnehmerbetrieb (engl.: *multiple access*), also z.B. den Zugriff mehrerer Mobilfunkteilnehmer auf ein Frequenzband. Allgemein lässt sich, wie u.a. in [16] beschrieben, zwischen spektralen, räumlichen, zeitlichen und codebasierten Multiplexverfahren unterscheiden, von denen die beiden erstgenannten Verfahren aufgrund der übergeordneten Relevanz im Kontext dieser Arbeit näher vorgestellt werden.

Frequenzmultiplex Der Frequenzmultiplex (engl.: *frequency division multiplex*, FDM) stellt das klassische Verfahren, u.a. im analogen Fernsprechnetz [56], zur Aufteilung in mehrere parallele Kanäle dar und wird zumeist für Vielfachnutzung (engl.: *frequency division multiple access*, FDMA) durch mehrere Teilnehmer genutzt. Hierbei werden mehrere zur Verfügung stehende Frequenzbänder verwendet, so dass eine frequenzversetzte Übertragung mehrerer Signale möglich ist. Aufgrund der allgemein

hohen benötigten Bandbreite für die Übertragung zeitdiskreter Signale wird FDMA i.d.R. nur bei analoger Nachrichtenübertragung eingesetzt.

OFDM ist ein Spezialfall des klassischen FDM-Verfahrens. Hierbei wird das Frequenzband in schmalbandige Unterbänder aufgeteilt. Die Spektren jedes Unterträgers überlappen sich dabei, wodurch Interferenzen (engl.: *inter carrier interference*, ICI) auftreten. Die durch Verwendung eines rechteckförmigen Sendefilters entstehenden si-Spektren lassen sich durch

$$S_k(f) = \frac{s_k}{|\Delta f|}\mathrm{si}\left(\frac{\pi f}{\Delta f}\right) \ , \tag{5.1}$$

mit $T_s = \frac{1}{\Delta f}$ als Symbolzeit für das Sendesignal s_k, ausdrücken [129]. Für $f = k\Delta f$ mit $k \in \mathbb{Z}^*$ ergibt sich $S(f_k) = 0$. Befinden sich die Maxima von si-Spektren weiterer Unterträger genau auf diesen Nulldurchgängen, liegt eine orthogonale Anordnung vor (siehe Abb. 5.2). Da sich die zugehörigen Signale untereinander nicht beeinflussen, erlaubt dieses Vorgehen im Idealfall die vollständige Rekonstruktion am Empfänger. Die so entstehenden Unterkanäle können i.d.R. als nicht-frequenzselektiv angesehen werden, d.h. es liegt keine zeitliche Überlagerung der Symbole vor.

Raummultiplex Die Anwendung des Raummultiplex (engl.: *space division multiplex*, SDM) basiert auf der räumlichen Trennung von Nutzdaten unter Verwendung mehrerer Antennen auf Sende- und/oder Empfangsseite. Durch eine feste Anordnung der Antennen ist es möglich, am Empfänger das gewünschte Signal herauszurechnen. Neben dem bereits angesprochenen MIMO-Verfahren, bei dem die Teilnehmer auf Sende- und Empfangsseite mehrere Antennen besitzen, existieren zudem Umsetzungen mit nur einer Antenne am Sender (engl.: *single input multiple output*, SIMO) oder am Empfänger (engl.: *multiple input single output*, MISO). Des Weiteren kann ein SDM-Übertragungssystem sende- oder empfangsseitig mehrere Teilnehmer besitzen (engl.: *multi user*, MU), die nur auf eine Untermenge der vorhandenen Antennen zugreifen.

5.1.3 Systemmodell

Wie vorangehend erwähnt sind Mehrantennensysteme und OFDM-Übertragungsverfahren im Rahmen dieser Arbeit von übergeordneter Bedeutung, da sie in vielen aktuellen Mobilfunkstandards verwendet werden [71]. Es soll daher im Folgenden ein schematischer Überblick über die jeweils notwendige Signalverarbeitung im Basisband gegeben werden. Darüber hinaus werden die Modellierung des Mobilfunkkanals sowie Auswirkungen möglicher Störungen auf die Empfangsqualität eingehend betrachtet.

Übertragungskanal Ein wichtiger Bestandteil eines Mobilfunksystems ist der Übertragungskanal, dessen Eigenschaften die empfangsseitige Rekonstruktion von Signalen maßgeblich beeinflussen [15]. U.a. durch Reflexion, Streuung und/oder Dämpfung kann das Signal stark verfälscht werden. Darüber hinaus müssen additive Störgrößen berücksichtigt werden, die z.B. durch Interferenzen anderer Nutzer oder thermische Effekte in Bauelementen auftreten können [56]. Im Rahmen dieser Arbeit wird von einem gedächtnisfreien Rayleigh-Modell als Kanal ausgegangen, d.h. es werden ausschließlich unabhängige, normalverteilte, mittelwertfreie und gaußverteilte Zufallszahlen verwendet. Des Weiteren soll im Kanal keine Leistung erzeugt werden. Die additive Störung wird durch ein gaußverteiltes weißes Rauschen (engl.: *additive white gaussian noise*, AWGN) modelliert. Unter Berücksichtigung dieser Effekte ergibt sich zunächst allgemein ein SISO- (engl.: *single input single output*) Systemmodell

$$y(t) = h(t) * s(t) + w(t) \ , \tag{5.2}$$

mit $*$ als Faltungsoperator, $h(t)$ als Impulsantwort des Kanals, $s(t)$ als gesendetes Signal und $w(t)$ als additives Rauschen, das als Grundlage für die nachfolgenden Ausführungen dient.

Mehrantennensystem Mehrantennensysteme werden zur Umsetzung des in Kap. 5.1.2 genannten SDM-Verfahrens angewendet. Am Empfänger müssen Einflüsse des Übertragungskanals sowie Rauscheffekte kompensiert bzw. minimiert werden, wofür viele unterschiedliche Detektionsverfahren existieren [11]. Durch den Einsatz eines Vorentzerrers kann der Kanaleinfluss am Sender herausgerechnet werden, was den Umfang der Signalverarbeitung im Empfänger häufig signifikant reduziert. Darüber hinaus erlaubt die Vorentzerrung die Umsetzung von Mehrantennensystemen, bei denen der Sender mit mehreren Antennen und empfangsseitig mehrere Teilnehmer mit z.B. jeweils nur einer Antenne ausgestattet sind (engl.: *multi user multiple intput single output*, MU-MISO), wenn die Empfänger nicht untereinander kommunizieren [88]. Abb. 5.3a zeigt den schematischen Aufbau einer MIMO-Übertragungsstrecke.

Die allgemeine Systemgleichung eines Mehrantennensystems lautet nach [11]

$$\mathbf{y}(t) = \mathbf{H}(t) * \mathbf{s}(t) + \mathbf{w}(t) \ , \tag{5.3}$$

mit $\mathbf{H}(t)$ als Kanalmatrix, $\mathbf{s}(t)$ als Symbolvektor und $\mathbf{w}(t)$ als Rauschvektor. Durch die Einschränkung des Systemmodells auf nicht-frequenzselektive und zeitinvariante Kanäle vereinfacht sich die Gl. (5.3) u.a. nach [92] zu

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{w} \ . \tag{5.4}$$

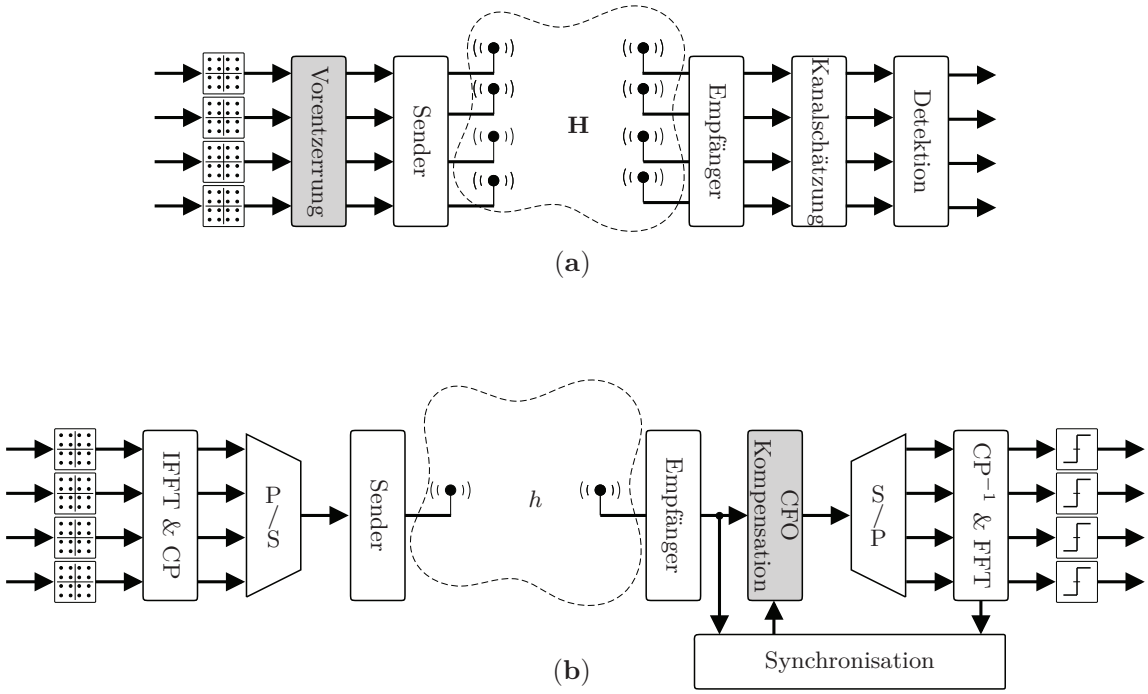


Abbildung 5.3: Aufbau der Basisband-Signalverarbeitung für **(a)** Mehrantennen- und **(b)** OFDM-Übertragungssysteme. Die Vorentzerrung stellt dabei einen optionalen Verarbeitungsschritt dar, der die empfangsseitige Detektion vereinfachen kann. Die grau unterlegten Blöcke markieren Module, die im Rahmen dieser Arbeit behandelt werden.

OFDM-System Wie in Kap. 5.1.2 erwähnt werden in einem OFDM-Symbol mehrere Daten parallel auf mehrere orthogonal angeordnete Unterträger aufgeteilt. U.a. nach [56] kann dies mathematisch durch Multiplikation mit einem Phasenterm beschrieben werden. Somit ergibt sich für das l-te OFDM-Symbol

$$s_{l,n} = \sum_{k=0}^{N-1} s_{l,k} e^{\frac{j2\pi kn}{N}} \ , \tag{5.5}$$

mit n bzw. k als Zeit- bzw. Frequenzindex und N als Anzahl der parallelen Daten. Dieser Ausdruck entspricht bis auf einen fehlenden Vorfaktor der Rechenvorschrift der inversen schnellen Fouriertransformation (engl.: *inverse fast fourier transform*, IFFT), weshalb die Bestimmung von $s_{l,n}$ als Transformation vom Frequenzbereich in den Zeitbereich angesehen werden kann. Zur Vermeidung von Symbolinterferenzen (engl.: *inter symbol interference*, ISI) muss zusätzlich ein zyklisches Schutzintervall (engl.: *cyclic prefix*, CP) eingefügt werden. Am Empfänger kann durch die Entfernung des CP sowie eine schnelle Fouriertransformation (engl.: *fast fourier transform*, FFT) das OFDM-Symbol zurückgewonnen werden. Ein graphischer Überblick einer zugehörigen Signalverarbeitungskette ist in Abb. 5.3b dargestellt.

Das OFDM-Systemmodell ist ausgehend von Gl. (5.2) im Zeitbereich durch

$$y_{l,n} = h_{l,n} * s_{l,n} + w_{l,n} \tag{5.6}$$

beschrieben. Durch die Transformation in den Frequenzbereich vereinfacht sich die Faltung zu einer Multiplikation.

Bitfehlerkurve Die Bewertung der Übertragungsqualität findet i.d.R. anhand von Bitfehlerkurven statt, bei denen das Auftreten von Bitfehlern (engl.: *bit error rate*, BER) über das normalisierte Träger-zu-Rausch-Verhältnis in $\frac{E_b}{N_0}$, mit E_b als Energie pro Bit und N_0 als Rauschleistung pro 1 Hz, angegeben wird [92].

5.2 Numerisch gesteuerter Oszillator

Ausgehend von den vorangehend beschriebenen Grundlagen werden im Folgenden ausgewählte Signalverarbeitungsschritte von Mehrantennen- bzw. OFDM-Systemen hinsichtlich performanter Implementierung elementarer Funktionen untersucht. Als Erstes werden hierfür numerisch gesteuerte Oszillatoren (engl.: *numerically controlled oscillator*, NCO), manchmal auch digitale Frequenzsynthese (engl.: *digital frequency synthesis*, DFS) genannt, betrachtet.

NCOs sind digitale Sinusgeneratoren, die durch Vorgabe eines Zahlenwerts eine feste Frequenz einstellen und z.B. in digitalen Phasenregelschleifen (engl.: *phase locked loop*, PLL) zum Einsatz kommen [34]. I.d.R. bestehen diese NCOs aus der Sinusfunktion und einem rückgekoppelten Akkumulator (siehe Abb. 5.4).

Zur Bewertung eines NCO-Hardwareentwurfs kann neben den allgemeinen Performance-Kriterien aus Kap. 2.4.1 auch die Genauigkeit des Funktionsverlaufs der Sinuskurve, ausgedrückt durch die Leistungsdifferenz der ersten Oberwelle zur Grundschwingung (engl.: *spurious free dynamic range*, SFDR), herangezogen werden [59]. Zudem lassen sich viele unterschiedliche Anwendungsgebiete, z.B. winkelbasierte Ortungsverfahren [7], erkennen, in denen Sinusgeneratoren verwendet werden. Im Bereich des Mobilfunks werden NCOs zur Frequenzoffsetkompensation eingesetzt, worauf im Folgenden detailliert eingegangen wird.

5.2.1 Einsatzgebiet im Mobilfunk

Im Bereich Mobilfunk werden NCOs zur Kompensation eines kontinuierlichen Trägerfrequenzversatzes (engl.: *carrier frequency offset*, CFO) Δf_{CFO} eingesetzt [16], der zu Fehlern bei der Rekonstruktion am Empfänger führen. Dabei können CFOs zum Einen durch den Dopplereffekt, der bei zeitlicher Variation der Abstands zwischen Sender und Empfänger auftritt, zum Anderen durch fertigungsbedingte Abweichungen des lokalen

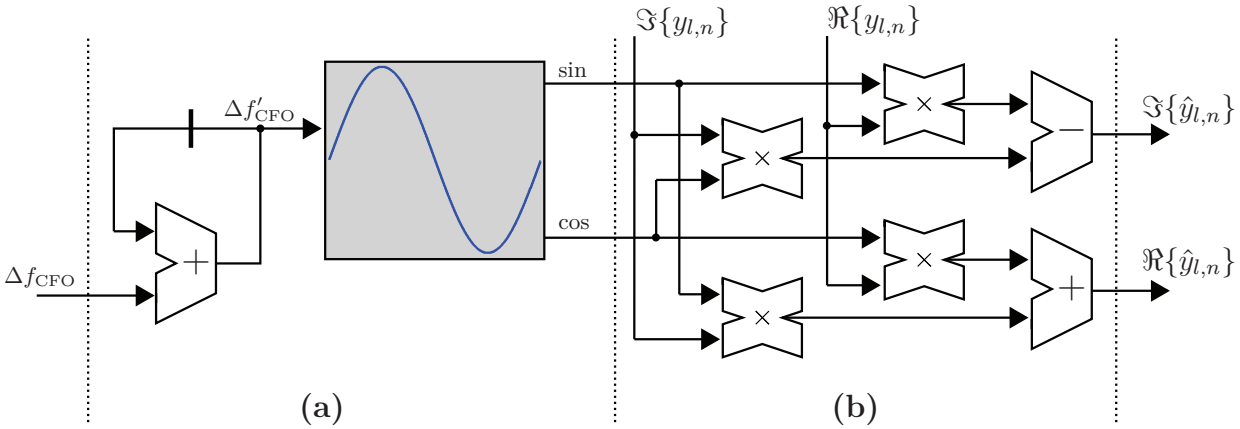


Abbildung 5.4: Hardwarearchitektur der CFO-Kompensation mit (**a**) dem NCO und (**b**) der Vektorrotation. Δf_{CFO} bezeichnet den Frequenzversatz, $\Re\{y_{l,n}\}$ bzw. $\Im\{y_{l,n}\}$ den Real- bzw. Imaginärteil des empfangenen Signals.

Oszillators (engl.: *local oscillators*, LO) entstehen. Durch die Synchronisation kann der Frequenzversatz Δf_{CFO} bestimmt werden; die Kompensation gleicht den Störeffekt aus.

Im Folgenden soll der CFO-Einfluss für das OFDM-Systemmodell aus Gl. (5.6) betrachtet werden, da OFDM-Systeme gegenüber Einträgersystemen deutlich anfälliger für CFOs sind [16]. Nach [128] ergibt sich im Zeitbereich

$$y_{l,n} = \mathrm{e}^{j2\pi \frac{\Delta f_{CFO} \cdot (n + l(N_{\mathrm{FFT}} + N_{\mathrm{CP}}))}{f_A}} (h_{l,n} * s_{l,n} + w_{l,n}) \ , \tag{5.7}$$

mit N_{FFT}, N_{CP}, n und f_A als Größe der FFT, Länge des CP, Eingangsindex der FFT und der Abtastfrequenz. Der Frequenzversatz ist also als Rotation im Zeitbereich interpretierbar, wodurch die CFO-Kompensation als Derotation durch

$$\hat{y}_{l,n} = y_{l,n} \mathrm{e}^{-j2\pi \frac{\Delta f_{CFO} \cdot (n + l(N_{\mathrm{FFT}} + N_{CP}))}{f_A}} \tag{5.8}$$

ausgedrückt werden kann. Die notwendige Rechenvorschrift zur CFO-Kompensation lautet somit für jeden Unterträger

$$\begin{pmatrix} \Re\{\hat{y}_{l,n}\} \\ \Im\{\hat{y}_{l,n}\} \end{pmatrix} = \begin{pmatrix} \Re\{y_{l,n}\} \cos \Delta f'_{CFO} + \Im\{y_{l,n}\} \sin \Delta f'_{CFO} \\ \Im\{y_{l,n}\} \cos \Delta f'_{CFO} - \Re\{y_{l,n}\} \sin \Delta f'_{CFO} \end{pmatrix} \ , \tag{5.9}$$

mit $\Delta f'_{CFO}$ als akkumulierter CFO sowie $\Re\{\centerdot\}$ und $\Im\{\centerdot\}$ als Real- und Imaginärteil. Eine entsprechende Hardwarearchitektur ist in Abb. 5.4 gezeigt. Zur effizienten Umsetzung ist somit eine effiziente Implementierung des Sinusgenerators notwendig, was im Folgenden durch ALFA-basierte Funktionsapproximation geschehen soll.

5.2.2 ALFA-Hardwarebeschleuniger

Aufgrund der vergleichsweise geringen algorithmischen Komplexität der CFO-Kompensation beschränkt sich die Umsetzung als Hardwarebeschleuniger fast ausschließlich auf die performante Umsetzung des Datenpfades [107]. Zur Speicherung von Zwischenergebnissen werden Register verwendet; der Einsatz eines Datenspeichermoduls ist nicht notwendig.

Datenpfad Der Datenpfad hat generell eine Wortbreite von 16 Bit ($Q.15$). In einigen Fällen werden nur 12 Bit ($Q.11$) benötigt, worauf im Folgenden explizit hingewiesen wird. Des Weiteren kann der Datenpfad in zwei unterschiedliche Bereiche unterteilt werden: NCO (siehe Abb. 5.4a) und Rotation (siehe Abb. 5.4b). Der NCO besteht aus einem Akkumulator und dem Sinusgenerator. Genau wie in Kap. 4.4.2 werden Symmetrieeffekte der Funktion ausgenutzt, um den Approximationsaufwand möglichst gering zu halten. So muss nur eine Viertelsinuswelle betrachtet werden, die weiteren Funktionsbereiche können durch Spiegelung ermittelt werden [115]. Zudem wird der Wertebereich auf $-1 \leq x \leq 1 - 2^{-15}$ skaliert. Für die Viertelsinuswelle ergeben sich somit 10 Bit. Die vollständige Umsetzung der skalierten Sinusfunktion lautet

$$\widetilde{\sin}(2\pi x) = \begin{cases} \widetilde{\sin}_{\frac{1}{4}}\left(2\pi x\right) & ;0 \geq x > \frac{1}{4} \\ \widetilde{\sin}_{\frac{1}{4}}\left(2\pi(\frac{1}{2} - x)\right) & ;\frac{1}{4} \geq x > \frac{1}{2} \\ -\widetilde{\sin}_{\frac{1}{4}}\left(2\pi(x - \frac{1}{2})\right) & ;\frac{1}{2} \geq x > \frac{3}{4} \\ -\widetilde{\sin}_{\frac{1}{4}}\left(2\pi(1 - x)\right) & ;\frac{3}{4} \geq x > 1 \end{cases}, \tag{5.10}$$

mit $\widetilde{\sin}_{\frac{1}{4}}$ als approximierte Viertelsinuswelle. Die Umkehrung des Vorzeichens vor dem Sinusterm kann durch Bildung des Einerkomplements mittels Negation vereinfacht berechnet werden. Die Subtraktion innerhalb der Sinusfunktion wird durch Negation und das Setzen bzw. Nicht-Setzen von MSBs realisiert. Mittels dieser Approximation kann, wie den Ergebnissen in Kap. 5.2.3 zu entnehmen ist, bereits für 12 Bit ($Q.11$) eine hinreichend hohe Rechengenauigkeit erreicht werden (siehe Abb. 5.6).

Neben der Implementierung der Sinusfunktion muss nach Gl. (5.9) auch die Kosinusfunktion für die CFO-Kompensation berücksichtigt werden. Dies kann durch die trigonometrische Beziehung

$$\widetilde{\cos}(2\pi x) = \widetilde{\sin}\left(2\pi(x + \frac{1}{4})\right) = \begin{cases} \widetilde{\sin}_{\frac{1}{4}}\left(2\pi(\frac{1}{2} - x)\right) & ;0 \geq x > \frac{1}{4} \\ -\widetilde{\sin}_{\frac{1}{4}}\left(2\pi(x - \frac{1}{2})\right) & ;\frac{1}{4} \geq x > \frac{1}{2} \\ -\widetilde{\sin}_{\frac{1}{4}}\left(2\pi(1 - x)\right) & ;\frac{1}{2} \geq x > \frac{3}{4} \\ \widetilde{\sin}_{\frac{1}{4}}\left(2\pi x\right) & ;\frac{3}{4} \geq x > 1 \end{cases}, \tag{5.11}$$

also eine Anpassung durch Modifikation der MSBs an die in Gl. (5.10) gegebenen Fallunterscheidung, geschehen.

Für die Vektorrotation werden nach Abb. 5.4b vier Multiplikationen und zwei Additionen benötigt. Die Multiplikation wird durch das in Kap. 3.3 angeführte Radixverfahren umgesetzt. Zur Komplexitätsreduktion soll der Eingang zur Ansteuerung der Multiplexer nur eine Wortbreite von 12 Bit besitzen, was den algorithmischen Signalverarbeitungsaufwand von acht auf sechs Addierer verringert. Der so entstehende reduzierte Radix-4 Multiplizierer (RRM) weist eine Abweichung der Rechengenauigkeit von

$$\mathrm{mae}_{RRM} = 2^{-13} \approx 1,22 \cdot 10^{-4} \tag{5.12}$$

auf.

Zur Reduktion des Energieverbrauchs werden die in Kap. 2.3 beschriebenen Implementierungstechniken *Clock-Gating* und *Operand-Isolation* berücksichtigt, die automatisch von der Logiksynthese in den Hardwareentwurf eingefügt werden. Zur Verringerung der Komplexität wird zudem auf das Zeitmultiplex-Verfahren zurückgegriffen: Durch entsprechende Aufteilung der Abarbeitungsreihenfolge ist es möglich, bei einer Latenz von fünf Takten die CFO-Kompensation durch sechs Addierer zu realisieren. Der RRM wird dabei in drei unterschiedliche Rechenphasen unterteilt: Die Berechnung der partiellen Produkte (RRM1) und zwei Phasen, in denen diese stellenwertrichtig akkumuliert werden (RRM2, RRM3) (siehe Abb. 5.5). Die sequenzielle Abarbeitung des Algorithmus sowie die zugehörige Verschaltung der Addierer ist im Folgenden beschrieben.

Steuerpfad Der Steuerpfad besteht aus einer FSM, die für die Berechnungsabfolge zuständig ist. Dies umfasst die Zuweisung der Operanden zu den zugehörigen Addierern und Registern. Die Abfolge der FSM wurde so festgelegt, dass die Hardware möglichst gleichmäßig ausgelastet ist. Ein Überblick der Abarbeitung ist sowohl in schriftlicher Form nachfolgend als auch graphisch aufbereitet in Abb. 5.5 dargestellt.

1. Im ersten Takt wird zunächst der approximierte Sinuswert berechnet, wobei ein Addierer für die eigentliche Funktionsapproximation, also QF= 1, sowie ein weiterer Addierer für den Akkumulator benötigt wird. Die übrigen Addierer werden für die Berechnung von RRM1 und RRM2 verwendet. Neben dem Sinuswert wird der Realteil des Empfangssignals mit dem Multiplizierer verbunden.

2. Im zweiten Takt werden zwei Addierer für die RRM3-Operation der im vorangehenden Takt begonnenen Multiplikation eingesetzt. Zudem werden wieder RRM1 und RRM2 berechnet, wobei nun der Imaginärteil mit dem Sinuswert multipliziert wird.

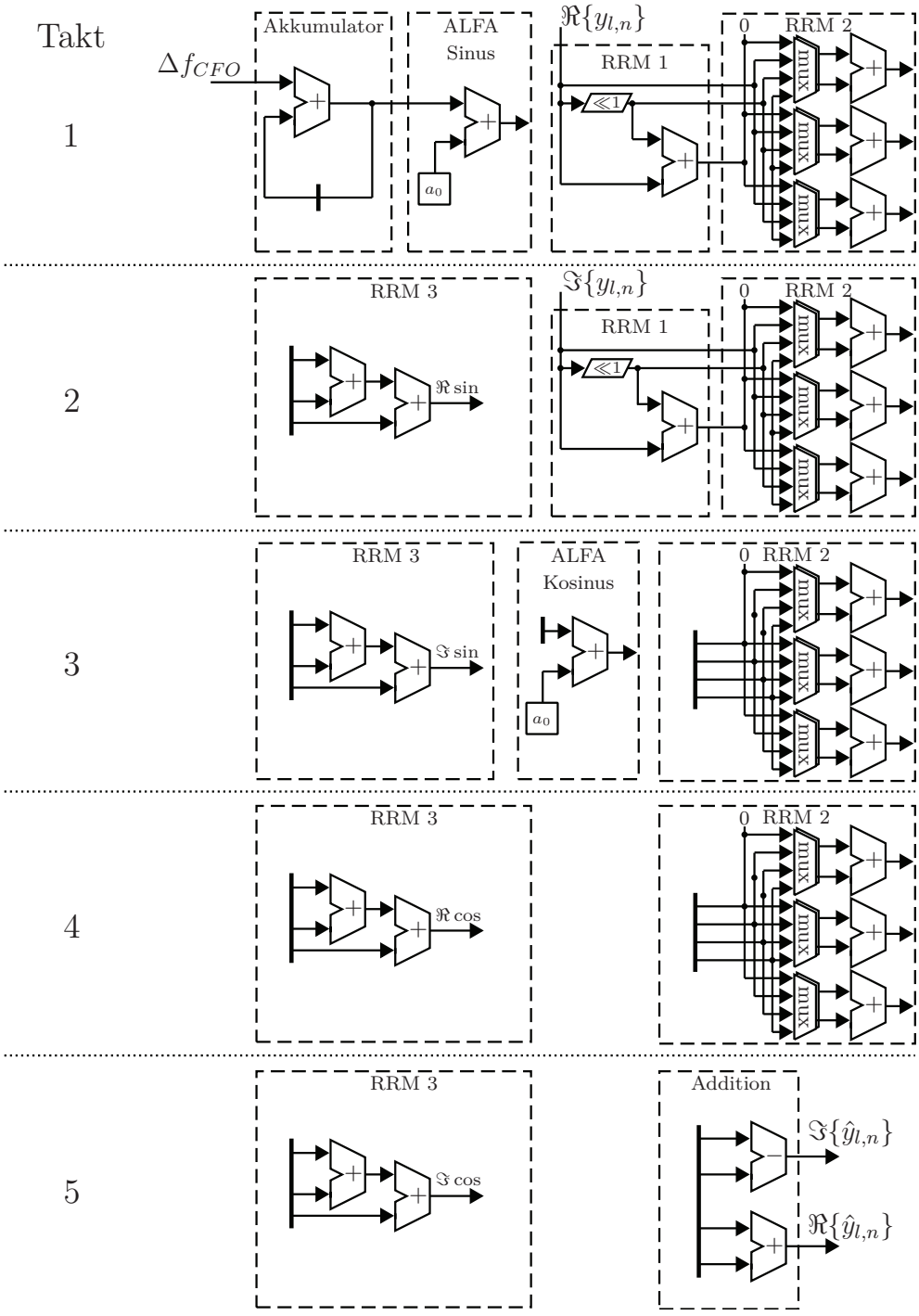


Abbildung 5.5: Schematischer Aufbau der ALFA-basierten CFO-Kompensation mit sechs Addierern im Zeitmultiplex und einer Latenz von fünf Takten. Akkumulation beschreibt die anfängliche Aufaddition der Frequenzversätze, ALFA Sinus/Kosinus benennt den approximierten Sinusgenerator, RRM1-3 die einzelnen Schritte der RRM4-Multiplikation sowie Addition den Abschluss der Vektorrotation. Die für die Radix-Multiplikation verwendeten Schiebeoperatoren zur stellenwertrichtigen Addition der Teilprodukte sind nicht angegeben.

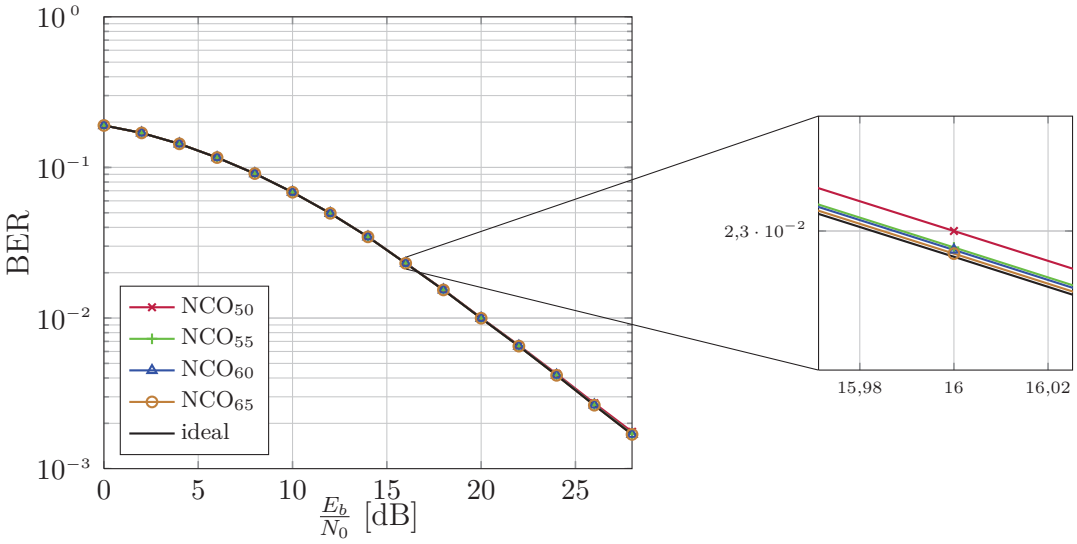


Abbildung 5.6: Bitfehlerkurve ALFA-basierter NCOs mit variierendem SFDR im Vergleich zur idealen, gleitkommabasierten Matlab-Umsetzung für ein 64-QAM-OFDM System.

3. Der dritte Takt beginnt wieder mit der Beendigung der bereits angefangenen Multiplikation. Darüber hinaus wird durch Gl. (5.11) der Kosinuswert ermittelt, der danach an die RRM2-Stufe weitergeleitet wird. Die Berechnungen der partiellen Produkte des Imaginär- und Realteils sind in den ersten beiden Takten bereits erfolgt und daher nicht mehr notwendig, weswegen RRM1 entfallen kann.

4. Der vierte Takt weist eine ähnliche Abfolge an Rechenschritten wie der zweite Takt auf, nur dass der Imaginärteil mit dem Kosinuswert multipliziert wird. Darüber hinaus entfällt wieder die RRM1-Berechnung, weswegen ein Addierer in diesem Takt ungenutzt bleibt.

5. Im fünften Takt wird die Berechnung der CFO-Kompensation durch die Berechnung von RRM3 und die beiden nachträglichen Additionen der Vektorrotation abgeschlossen. Somit bleiben in diesem Takt zwei Addierer ungenutzt.

5.2.3 Evaluation

Zur Evaluation des ALFA-basierten Hardwarebeschleunigers zur CFO-Kompensation werden sowohl unterschiedliche Rechengenauigkeiten der Funktionsapproximation betrachtet und anwendungsspezifisch bewertet als auch eine zugehörige ASIC-Implementierung erstellt, weswegen beides im Folgenden vorgestellt wird.

Tabelle 5.1: ALFA-basierte NCO-Modellierung mit unterschiedlichen SFDR-Werten.

Referenz	QF	SFDR	mae_{ALFA}	Segmente
NCO_{50}	1	$52,8$	$7,32 \cdot 10^{-3}$	13
NCO_{55}	1	$58,6$	$5,73 \cdot 10^{-3}$	19
NCO_{60}	1	$61,6$	$3,42 \cdot 10^{-3}$	26
NCO_{65}	1	$66,1$	$1,47 \cdot 10^{-3}$	63

Rechengenauigkeit Die Rechengenauigkeit wird für die CFO-Kompensation anhand des SFDR-Werts beurteilt [59]. Mehrere ALFA-basierte Approximationen mit 12 Bit Datenwortbreite sowie unterschiedlicher Genauigkeit NCO_{50}, NCO_{55}, NCO_{60} bzw. NCO_{65}, die einen SFDR-Wert von 50 dBc, 55 dBc, 60 dBc bzw. 65 dBc besitzen, werden dabei untersucht. Die Ergebnisse sind in Tab. 5.1 zusammengefasst. Des Weiteren wird die Rechengenauigkeit der ALFA-basierten CFO-Kompensation simulativ anhand eines entsprechenden Matlab-Modells überprüft. Das Kanalmodell besitzt die in Kap. 5.1.3 beschriebenen Eigenschaften und ist empfangsseitig bekannt. Für das Übertragungsmodell wird auf eine gängige LTE-Konfiguration (SISO-OFDM) mit 64-QAM, einer FFT-Größe von 128 bei 72 genutzten Unterträgern, $f_A = 1,92$ MHz und ein CP der Länge $8,33\mu$s zurückgegriffen [131]. Der Abstand der Unterträger ist 15 kHz; der maximale Frequenzversatz beträgt $\Delta f_{CFO} = 7,5$ kHz. Zur Auswertung wird die BER-Kurve nach der ML-Detektion [11] betrachtet und mit den Ergebnissen einer idealen Matlab-basierten CFO-Kompensation verglichen. Alle ALFA-basierten Ansätze weichen, wie in Abb. 5.1 gezeigt, nur geringfügig von der idealen BER-Kurve ab, weswegen die Approximation mit der geringsten Segmentanzahl und somit auch der geringsten Schaltungskomplexität (NCO_{50}) für den nachfolgenden Implementierungsschritt verwendet wird.

Implementierung Für die Implementierung der vorgestellten CFO-Kompensation wird ausgehend vom Matlab-Modell, das als Spezifikation und zur Verifikation verwendet wird, eine äquivalente Hardwarebeschreibung in VHDL erzeugt. Die Approximation der Sinusfunktion wird dabei mittels ALFA-Synthese erstellt. Durch Logik- und Layoutsynthese wird das Layout erzeugt. Als Fertigungsprozess wird die 130nm Faraday Technologie der Firma UMC verwendet [119]. Im Vergleich zu aktuellen Referenzen erzielt die ALFA-Hardwarearchitektur zur CFO-Kompensation sehr gute Ergebnisse hinsichtlich Komplexität und Energie- bzw. Leistungsverbrauch (siehe Tab. 5.2).

Die Evaluation zeichnet den ALFA-basierten NCO als performanten Ansatz im Mobilfunk aus. Da die Derotation bereits für eine geringe Rechengenauigkeit ausreichend präzise Ergebnisse liefert, können entsprechend unkritische Vorgaben für die ALFA-Synthese formuliert werden. Der Quantisierungsfaktor von Eins erlaubt effizienten Zeit-

Tabelle 5.2: Syntheseergebnisse des ALFA-Hardwarebeschleunigers zur CFO-Kompensation (fett gedruckt) verglichen mit aktuellen Referenzen. Die Komplexität wird neben der üblichen Angabe in Gatteräquivalenten zudem als normierte Fläche nach [4] angegeben, die sich als Quotient aus Chipfläche durch die quadrierte Prozessgröße berechnet.

Referenz	Prozessgröße [nm]	Frequenz [MHz]	Latenz [Takte]	Komplexität [kGE $/10^5$]	Leistungsverbrauch [mW]
NCO_{50}	**130**	**181**	**5**	**9381/20,16**	**3,37**
[28]	130	132	4	-/94,60	8,40
[22]	250	385	5	-/35,20	154

multiplex der verwendeten Addierer und begrenzt die vollständige Abarbeitung der Derotation auf fünf Takte. Neben den bereits angesprochenen performanten Ergebnissen im Bereich Energieverbrauch und Komplexität führt dies auch zu guten Werten hinsichtlich Taktfrequenz und Latenz: So ist die Berechnung einer CFO-Kompensation mit einer Taktfrequenz von $36,2$ MHz möglich, was z.B. für LTE ausreichend schnell ist.

5.3 QR-Zerlegung

Die QR-Zerlegung ist ein algorithmisches Verfahren zur Zerlegung einer gegebenen Matrix $\mathbf{A}$ in eine unitäre Matrix $\mathbf{Q}$ und eine obere Dreiecksmatrix $\mathbf{R}$ nach der Rechenvorschrift

$$\mathbf{A} = \mathbf{QR} \ . \tag{5.13}$$

Sie kommt in vielen unterschiedlichen mathematischen Bereichen zum Einsatz, z.B. beim Lösen von Gleichungssystemen [36]. Im Mobilfunk lassen sich bei Mehrantennensystemen mehrere Anwendungsgebiete angeben (siehe Kap. 5.3.2). Für $m \times n$ Matrizen mit $\mathbf{A} \in \mathbb{C}^{m \times n}; m \geq n$ kann die QR-Zerlegung nach

$$\mathbf{A} = \mathbf{QR} = (\mathbf{Q}_1 \mathbf{Q}_2) \begin{pmatrix} \mathbf{R}_1 \\ \mathbf{0}_{(m-n)\times n} \end{pmatrix} = \mathbf{Q}_1 \mathbf{R}_1 \tag{5.14}$$

mit $\mathrm{Q}_1 \in \mathbb{C}^{m \times n}$, $\mathbf{Q}_2 \in \mathbb{C}^{m \times (m-n)}$ und $\mathbf{R}_1 \in \mathbb{C}^{n \times n}$, sowie m bzw. n als Anzahl an Empfangs- bzw. Sendeantennen, ermittelt werden. Nachfolgend wird ausschließlich von quadratischen und regulären Matrizen $\mathbf{A} \in \mathbb{C}^{n \times n}$ ausgegangen.

5.3.1 Algorithmen

Die QR-Zerlegung lässt sich durch eine Vielzahl unterschiedlicher Algorithmen berechnen [36]. Für die HDL-Implementierung existieren zwei effiziente Ansätze, die im Folgenden vorgestellt werden.

***Modified Gram-Schmidt*-Verfahren** Die Bestimmung der QR-Zerlegung durch das *Modified Gram-Schmidt*-Verfahren basiert auf der Berechnung von $\mathbf{Q}$ und $\mathbf{R}$ durch spaltenweise Orthonormalisierung einer Matrix. Hierfür wird zunächst die $\mathbf{Q}$-Matrix durch Orthogonalisierung der Spalten nach

$$\mathbf{q}'_i = \mathbf{a}_i - \sum_{j=1}^{i-1} \frac{\langle \mathbf{q}'_j, \mathbf{a}'_i \rangle}{\langle \mathbf{q}'_j, \mathbf{q}'_j \rangle} \mathbf{q}'_j \ , \tag{5.15}$$

mit $\mathbf{q}_i$ bzw. $\mathbf{a}_i$ als i-te Spalte von $\mathbf{Q}$ bzw. $\mathbf{A}$, durchgeführt. In einem zweiten Schritt werden die Vektoren durch

$$\mathbf{q_i} = \frac{\mathbf{q}_i}{||\mathbf{q}_i||_2} \ , \tag{5.16}$$

mit $||\ \mathbf{.}\ ||_2$ als L^2-Norm, berechnet. Die obere Dreiecksmatrix R kann für komplexe Matrizen abschließend durch

$$\mathbf{R} = \mathbf{Q}^H \mathbf{A} \tag{5.17}$$

bestimmt werden,

Givens-Rotation Bei der Givens-Rotation wird die QR-Zerlegung durch Multiplikation mit einer Rotationsmatrix

$$\mathbf{G} = \begin{pmatrix} 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \dots & \cos(\varphi) & -\sin(\varphi) & \dots & 0 \\ 0 & \dots & \sin(\varphi) & \cos(\varphi) & \dots & 0 \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & \dots & 1 \end{pmatrix} \tag{5.18}$$

$$\mathbf{H} = \begin{pmatrix} \mathbb{C} & \mathbb{C} & \mathbb{C} \\ \mathbb{C} & \mathbb{C} & \mathbb{C} \\ \mathbb{C} & \mathbb{C} & \mathbb{C} \end{pmatrix} \rightarrow \begin{pmatrix} \mathbb{R} & \mathbb{C} & \mathbb{C} \\ \mathbb{R} & \mathbb{C} & \mathbb{C} \\ \mathbb{R} & \mathbb{C} & \mathbb{C} \end{pmatrix} \rightarrow \begin{pmatrix} \mathbb{R} & \mathbb{C} & \mathbb{C} \\ 0 & \mathbb{C} & \mathbb{C} \\ 0 & \mathbb{C} & \mathbb{C} \end{pmatrix} \rightarrow \begin{pmatrix} \mathbb{R} & \mathbb{C} & \mathbb{C} \\ 0 & \mathbb{R} & \mathbb{C} \\ 0 & \mathbb{R} & \mathbb{C} \end{pmatrix} \rightarrow \begin{pmatrix} \mathbb{R} & \mathbb{C} & \mathbb{C} \\ 0 & \mathbb{R} & \mathbb{C} \\ 0 & 0 & \mathbb{C} \end{pmatrix} \rightarrow \begin{pmatrix} \mathbb{R} & \mathbb{C} & \mathbb{C} \\ 0 & \mathbb{R} & \mathbb{C} \\ 0 & 0 & \mathbb{R} \end{pmatrix} = \mathbf{R}$$

(a)

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} \mathbb{C} & 0 & 0 \\ 0 & \mathbb{C} & 0 \\ 0 & 0 & \mathbb{C} \end{pmatrix} \rightarrow \begin{pmatrix} \mathbb{C} & \mathbb{C} & \mathbb{C} \\ \mathbb{C} & \mathbb{C} & 0 \\ \mathbb{C} & \mathbb{C} & \mathbb{C} \end{pmatrix} \rightarrow \begin{pmatrix} \mathbb{C} & \mathbb{C} & \mathbb{C} \\ \mathbb{C} & \mathbb{C} & 0 \\ \mathbb{C} & \mathbb{C} & \mathbb{C} \end{pmatrix} \rightarrow \begin{pmatrix} \mathbb{C} & \mathbb{C} & \mathbb{C} \\ \mathbb{C} & \mathbb{C} & \mathbb{C} \\ \mathbb{C} & \mathbb{C} & \mathbb{C} \end{pmatrix} \rightarrow \begin{pmatrix} \mathbb{C} & \mathbb{C} & \mathbb{C} \\ \mathbb{C} & \mathbb{C} & \mathbb{C} \\ \mathbb{C} & \mathbb{C} & \mathbb{C} \end{pmatrix} = \mathbf{Q}^H$$

(b)

Abbildung 5.7: Beispielhafte Abarbeitungssequenz der Givens-Rotation zur Berechnung der QR-Zerlegung anhand einer komplexen 3×3 Matrix. (**a**) $\mathbf{R}$ kann dabei direkt aus der Kanalmatrix $\mathbf{H}$, (**b**) $\mathbf{Q}$ aus der Einheitsmatrix $\mathbf{I}$ abgeleitet werden. Orange bzw. blau unterlegte Elemente stellen die Erzeugung der reellen Zahlen bzw. Nulleinträge dar, hellorange bzw. hellblaue Elemente die jeweils zugehörige Rotation (siehe Kap. 5.3.1).

durch Drehung in der Ebene um den Winkel φ, berechnet. Vereinfacht dargestellt ergibt sich die zweidimensionale Rechenvorschrift

$$\begin{pmatrix} x'_n \\ x'_{n+1} \end{pmatrix} = \mathbf{G}' \begin{pmatrix} x_n \\ x_{n+1} \end{pmatrix} \quad ; \quad \mathbf{G}' = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{pmatrix} , \tag{5.19}$$

mit $\mathbf{x}$ und $\mathbf{x}'$ als ursprünglicher und gedrehter Vektor sowie i als entsprechender Zeilenindex. Für die QR-Zerlegung wird mittels Givens-Rotation eine obere Dreiecksmatrix aus $\mathbf{A}$ erzeugt, wofür die Matrixeinträge nacheinander auf Null gedreht werden. Die Matrix $\mathbf{Q}$ ergibt sich aus der Anwendung der Vektorrotation auf die Einheitsmatrix $\mathbf{I}$. Die QR-Zerlegung durch Givens-Rotation besteht somit zum Einen aus der Berechnung der Sinus- und Kosinuswerte von $\mathbf{G}$, nachfolgend Erzeugung genannt, zum Anderen aus der Drehung von Vektoren durch $\mathbf{G}'$, nachfolgend Rotation genannt. Eine gängige Abarbeitungssequenz einer Givens-Rotation-basierten QR-Zerlegung ist in Abb. 5.7 gezeigt.

5.3.2 QR-Zerlegung im Mobilfunk

Im Bereich Mobilfunk findet die QR-Zerlegung für Mehrantennensysteme für die Detektion oder (Vor-)Entzerrung Anwendung. Dabei wird sie i.d.R. auf die komplexe Kanalmatrix mit

$$\mathbf{H} = \mathbf{QR} \tag{5.20}$$

angewendet. Dazu ist die vollständige Kenntnis der Kanalimpulsantwort notwendig, die im Empfänger z.B. durch das Senden fest vorgegebener Symbole, sogenannte Piloten, geschätzt werden kann. Im Sender kann $\mathbf{H}$ durch Rückkopplung oder das Ausnutzen von Kanalreziprozitätseffekten ermittelt werden [12]. Zwei gängige Anwendungsgebiete der QR-Zerlegung im Mobilfunk sind nachfolgend beschrieben.

Detektion Zentrale Aufgabe von Detektoren im Mobilfunk ist es, die gesendeten Symbole aus dem Empfangsvektor zu ermitteln. Zur Umsetzung in Mehrantennensystemen lassen sich nach [11] eine Vielzahl unterschiedlicher Ansätze angeben, die eine genäherte Lösung für das in Gl. (5.3) gegebene Gleichungssystem berechnen. Zum Einen kann eine QR-Zerlegung als Vorverarbeitungsschritt für nichtlineare Detektoren, z.B. *Sphere-Decoder* [101] oder *K-Best* [43], verwendet werden, um das Problem in einzelne, eindimensionale Schritte zu zerlegen. Zum Anderen kann z.B. für lineare Entzerrung die Pseudoinverse mittels QR-Zerlegung bestimmt werden. Das grundlegende Prinzip basiert dabei auf der in Kap. 3.5.1 eingeführten KQ-Methode. Je nach Detektor wird anhand dieser Zerlegung eine geschätzte Lösung für den Symbolvektor $\mathbf{s}$ errechnet.

Vorentzerrung Das grundlegende Prinzip der Vorentzerrung für Mehrantennensysteme basiert auf der Verlagerung einzelner Rechenschritte des Empfängers zum Sender [130]. So wird der Einfluss der Kanalverzerrung vor der eigentlichen Übertragung in den Sendedaten berücksichtigt, wodurch der empfangsseitige Rechenaufwand der Detektion deutlich reduziert werden kann. Zur Umsetzung lassen sich unterschiedliche mathematische Ansätze angeben; im Rahmen dieser Arbeit wird die Tomlinson-Harashima-Vorentzerrung (engl: *Tomlinson-Harashima precoder*, THP) mit *Zero-Forcing* (ZF) betrachtet.

Das grundlegende Prinzip von THP basiert auf einer entscheidungsrückgekoppelten Entzerrung (engl.: *decision feedback equalization*, DFE), bei der die Kanalmatrix durch Multiplikation mit einer orthonormalen Matrix in eine Dreiecksmatrix zerlegt wird [92]. Durch QR-Zerlegung ergibt sich zunächst

$$\mathbf{R} = \mathbf{Q}^H \mathbf{H} \, . \tag{5.21}$$

Zur Vorentzerrung wird die DFE zum Sender verschoben. Durch Rückkopplung können für jeden räumlich getrennten Kanal die Einflüsse der übrigen Kanäle minimiert werden. Dies kann durch

$$\mathbf{B} = \mathbf{H}^H \mathbf{Q} \mathbf{D}^{-1} - \mathbf{I} = \mathbf{R}^H \mathbf{D}^{-1} - \mathbf{I} = \mathbf{L} \mathbf{D}^{-1} - \mathbf{I} \, , \tag{5.22}$$

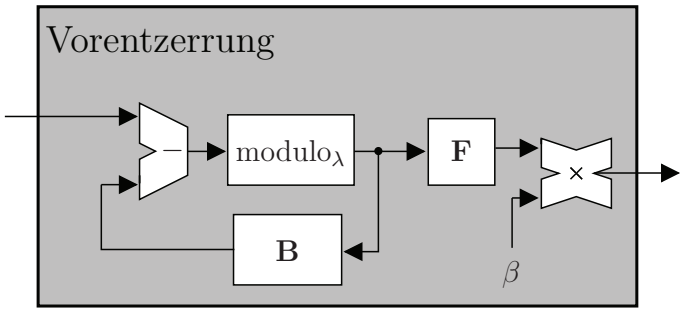


Abbildung 5.8: Prinzipieller Aufbau der sendeseitigen THP-ZF Vorentzerrung mit **B** als Feedbackschleife, **F** als Sendefilter und β als Skalierungsfaktor.

mit **I** als Einheitsmatrix und **L** als linke untere Dreiecksmatrix umgesetzt werden. **D** dient der Normierung der Sendeleistung und ist durch

$$\mathbf{D} = \mathrm{dg}(\mathbf{R}) = \begin{pmatrix} r_{1,1} \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots r_{n,n} \end{pmatrix} \tag{5.23}$$

definiert. Das Sendefilter ergibt sich zu

$$\mathbf{F} = \mathbf{L}\mathbf{D}^{-\mathbf{1}} \ . \tag{5.24}$$

Da der Sendevektor durch Anwendung der Vorentzerrung weit außerhalb des definierten Signalraums liegen kann, wird das entsprechende Symbol mittels spezieller Modulo-Operation zurück in diesen Bereich, die so genannte fundamentale Voronoi-Region, abgebildet, was die Sendeleistung begrenzt. Es gilt wie u.a. in [38] gegeben

$$\mathrm{modulo}_\lambda(x) = x - \left\lfloor \frac{x}{\lambda} + \frac{1}{2} \right\rfloor \lambda \ \ ; \ \lambda = 2\sqrt{M} \ , \tag{5.25}$$

mit M als Anzahl der Signalzuordnungen (siehe Kap. 5.1.1), womit x auf einem festen Punkt im Bereich $[-\lambda/2, \lambda/2)$ liegt. Durch eine empfängerseitige Modulo-Operation wird das Symbol an die ursprüngliche Position zurückgesetzt.

Zur Beschränkung der mittleren Sendeenergie wird des Weiteren ein zusätzlicher Skalierungsfaktor β verwendet, der sowohl sende- als auch empfängerseitig berücksichtigt werden muss. Weitergehende Erläuterungen dazu sowie zur Vorentzerrung allgemein finden sich u.a. in [130]. Der sendeseitige Aufbau der THP-ZF-Vorentzerrung ist in Abb. 5.8 dargestellt.

5.3.3 ALFA-Hardwarebeschleuniger

Zur ALFA-basierten Implementierung der QR-Zerlegung soll ein Hardwarebeschleuniger eingesetzt werden, der auf der Givens-Rotation als zugrunde liegendem Algorithmus basiert [100]. Die Erzeugung der benötigten Sinus- und Kosinusglieder kann dabei nach den Rechenvorschriften

$$\cos(\varphi) = \frac{x_i}{\sqrt{x_i^2 + x_{i+1}^2}} \quad \text{und} \tag{5.26}$$

$$\sin(\varphi) = -\frac{x_{i+1}}{\sqrt{x_i^2 + x_{i+1}^2}} \,, \tag{5.27}$$

mit x_i und x_{i+1} als Element an der i bzw. $i+1$-ten Stelle, umgesetzt werden. Zusammen mit Gl. (5.19) für die Anwendung der Givens-Rotation kann die QR-Zerlegung vollständig berechnet werden.

Bei näherer Betrachtung der Gleichungen fällt auf, dass alle notwendigen Rechenschritte durch Basisoperatoren entweder im Festkomma- oder logarithmischen Zahlenformat berechnet werden können, weshalb die Implementierung entsprechend realisiert werden soll (siehe Kap. 3.1.3). Wie in Kap. 4.4.2 werden zur Transformation der Zahlenformate die Konvertermodule LOG(x) und ALOG(x) für die Abbildung der nichtlinearen Funktionen $\mathrm{ld}(x+1)$ und $2^x - 1$ verwendet, die durch ALFA-Synthese erstellt werden. Der prinzipielle Aufbau des Hardwarebeschleunigers besteht aus einem Speicher sowie dem Daten- und Steuerpfad, die im Folgenden näher erläutert werden. Abb. 5.9 zeigt die dazugehörige Struktur.

Speicher Der Datenspeicher ist an ein 4×4 Mehrantennensystem, also eine komplexwertige 4×4 Matrix angepasst, was eine Gesamtanzahl von 64 Speicherstellen ergibt, die durch Register realisiert sind. Die Datenwortbreite beträgt, da unterschiedliche Rechengenauigkeiten verwendet werden, 18 bzw. 16 Bit. Aufgrund der Umsetzung des Datenpfades werden die Werte ausschließlich im logarithmischen Zahlenformat im Speicher hinterlegt.

Datenpfad Als Zahlenformat wird für den Datenpfad in Festkommadarstellung das $Q2.15$- bzw. $Q2.13$-Format, für die logarithmische Zahlendarstellung 5 Bit für den Exponenten, 9 bzw. 11 Bits für die Mantisse, sowie jeweils ein Bit für Vorzeichen und den Nullwert gewählt. Zur Implementierung wird der Datenpfad in die Erzeugung der Sinus- und Kosinusglieder, nachfolgend Erzeugung genannt, und die Rotation unterteilt. Für Ersteres werden zur Vermeidung von Überläufen Gl. (5.26) und Gl. (5.27) in

$$\cos(\varphi) = \begin{cases} \frac{1}{\sqrt{1+\left(\frac{x_{i+1}}{x_i}\right)^2}} & ;\ x_i > x_{i+1} \\ \frac{\frac{x_i}{x_{i+1}}}{\sqrt{1+\left(\frac{x_i}{x_{i+1}}\right)^2}} & ;\ \text{sonst} \end{cases} \tag{5.28}$$

und

$$\sin(\varphi) = \begin{cases} \frac{\frac{x_{i+1}}{x_i}}{\sqrt{1+\left(\frac{x_{i+1}}{x_i}\right)^2}} & ;\ x_i > x_{i+1} \\ \frac{1}{\sqrt{1+\left(\frac{x_i}{x_{i+1}}\right)^2}} & ;\ \text{sonst} \end{cases} , \tag{5.29}$$

mit i als Zeilenindex, transformiert. Des Weiteren vereinfacht diese Notation die Addition im Nenner auf das Setzen eines Bits auf Eins, da der Ausdruck $\left(\frac{x_i}{x_{i+1}}\right)$ bzw. $\left(\frac{x_{i+1}}{x_i}\right)$ immer kleiner als Eins ist. Die Rotation der ersten Spalte kann zudem durch

$$r = \begin{cases} x_i\sqrt{1+\left(\frac{x_{i+1}}{x_i}\right)^2} & ;\ x_i > x_{i+1} \\ x_{i+1}\sqrt{1+\left(\frac{x_i}{x_{i+1}}\right)^2} & ;\ \text{sonst} \end{cases} \tag{5.30}$$

berechnet werden. Aufgrund der ähnlichen mathematischen Gleichungen von $\sin(\varphi)$, $\cos(\varphi)$ und r kann ein gemeinsamer Datenpfad hierfür verwendet werden. Die Rotation ergibt sich durch direkte Umsetzung von Gl. (5.19).

Zur Steigerung der Performance werden unterschiedliche Implementierungstechniken aus Kap. 2.3 verwendet, wobei die Minimierung der Komplexität im Vordergrund steht. Zunächst wird der kritische Pfad durch das Einführen von *Pipeline*-Registern reduziert. Des Weiteren werden durch Anwendung des Zeitmultiplex-Verfahrens nur vier Addierer im Datenpfad benötigt. Somit lässt sich die Erzeugung in zwei, die Rotation in drei Takten berechnen. Ein Überblick über den Aufbau des Datenpfades ist in Abb. 5.9 gezeigt.

Zur Verringerung der Latenz und zur Vermeidung unerlaubter Divisionen wird das Auftreten von Nullen gesondert behandelt: Ist bei der Erzeugung $x_{i+1} = 0$, werden sämtliche Zeileneinträge übersprungen. Bei der Rotation werden Vektoren, die ausschließlich Nulleinträge besitzen, übersprungen.

Steuerpfad Der Steuerpfad besteht hauptsächlich aus einer zählerbasierten FSM, die für die Abarbeitung der QR-Zerlegung zuständig ist. Je nach aktuellem Zustand wird das Inkrement dabei angepasst. Ähnlich den in Kap. 4.4.1 vorgestellten Schaltungen ist dies auf zwei Hierarchieebenen aufgeteilt: Zum Einen findet eine lokale Ansteuerung

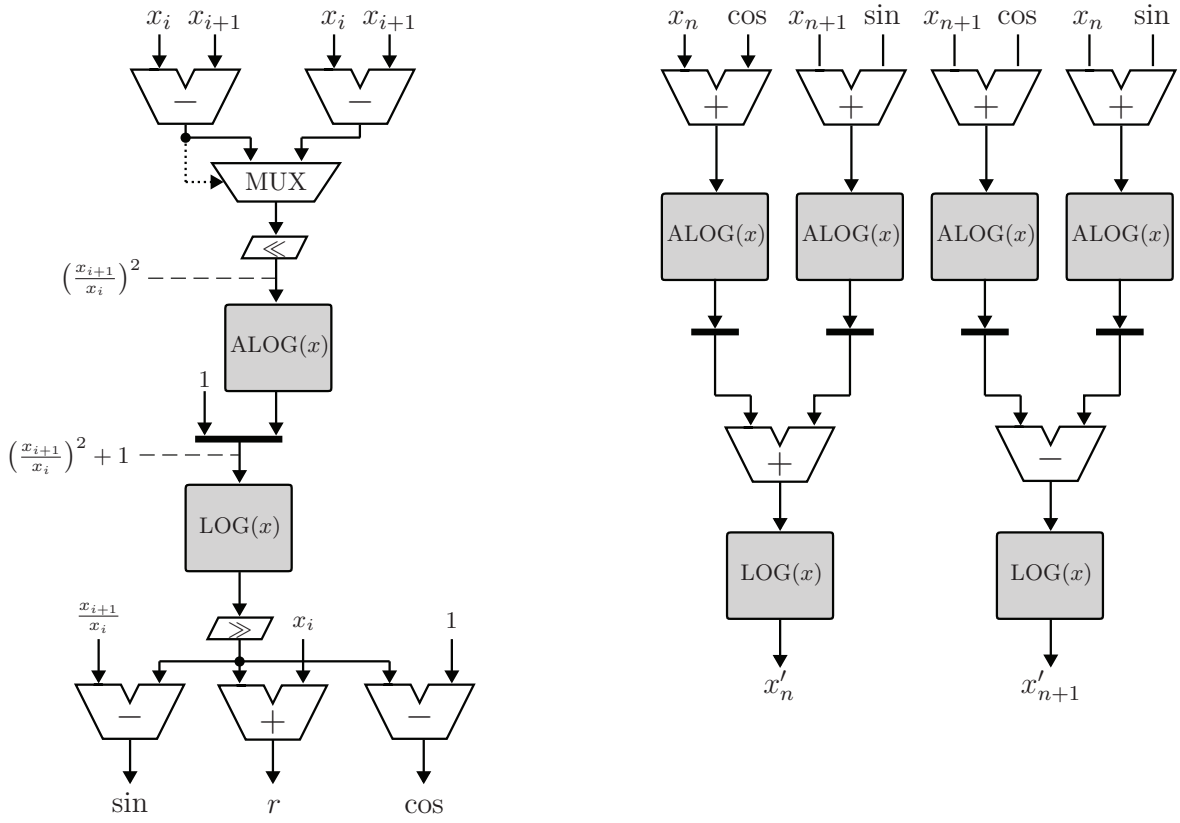


Abbildung 5.9: Datenpfad-Architektur der ALFA-basierten QR-Zerlegung mittels logarithmischen Zahlenformats mit (**a**) der Erzeugung und (**b**) der Rotation. Die gepunktete Linie markiert die Ansteuerung des Multiplexers, mit dessen Hilfe der Quotient kleiner eins ausgewählt wird. Die gestrichelten Linien zeigen Zwischenergebnisse für $x_i > x_{i+1}$ an.

statt, bei der die Operanden des Datenpfades adressiert werden sowie der Datenpfad konfiguriert wird. Dabei können die Adressen direkt aus dem aktuellen Zustand der FSM abgelesen werden. Zum Anderen wird eine globale Ansteuerung berücksichtigt, die für die sequenzielle Abarbeitung der QR-Zerlegung zuständig ist. Da die Einträge der Kanalmatrix **H** komplex sind, muss die Givens-Rotation entsprechend angepasst werden. So werden in einem ersten Schritt die komplexen Werte der betrachteten Spalteneinträge auf reelle Werte gedreht. Danach findet die Drehung der Werte auf Null statt. Die Abarbeitungsreihenfolge entspricht dabei dem in Abb. 5.7 gegebenen Schema.

5.3.4 Evaluation

Zur Evaluation der ALFA-basieren Implementierung der QR-Zerlegung werden drei unterschiedliche Aspekte betrachtet: Zunächst findet eine Auswertung der algorithmischen Komplexität statt. Darauf folgt die Analyse der Rechengenauigkeit anhand

Tabelle 5.3: Rechenkomplexität des ALFA-basierten Datenpfades mit QF= 1 im Vergleich zu CORDIC-basierten Umsetzungen mit m Iterationsschritten. Die algorithmische Latenz bezeichnet die längste Sequenz an Addierern, die im jeweiligen Entwurf verwendet wird.

	CORDIC_m	ALFA
Algorithmische Latenz	m	8
Anzahl Addierer Erzeugung	$2m$	7
Anzahl Addierer Rotation	$2m$	12

der Verwendung in einem wie in Kap. 5.3.2 eingeführten THP-ZF Vorentzerrers. Den Abschluss der Evaluation stellt die Auswertung der Implementierung dar, bei der ein Schaltungslayout erstellt wird.

Algorithmenkomplexität Für die Auswertung der algorithmischen Komplexität werden die benötigten mathematischen Operationen betrachtet. Auch hierbei werden Erzeugung und Rotation gesondert voneinander behandelt. Die Bewertung der Ergebnisse findet zudem anhand einer CORDIC-Umsetzung mit gleicher Funktion statt (siehe Kap. 3.4.2), welche die Erzeugung und Rotation vollständig mit einer CORDIC-Operation berechnen kann [113]. Aufgrund der Beschaffenheit des CORDIC kann die Rechengenauigkeit in Abhängigkeit von Signalverarbeitungskomplexität durch die Anzahl der Iterationsschritte variieren. Für den ALFA-basierten Ansatz kann der Rechenaufwand durch Vorgabe des Quantisierungsfaktors gesteuert werden und wird zunächst auf Eins gesetzt. Tab. 5.3 zeigt einen Überblick über die jeweiligen Ergebnisse beider Ansätze hinsichtlich der Anzahl an Addierern, der einzigen verwendeten Operationen, und algorithmischer Latenz, die in diesem Zusammenhang als längste Sequenz zu durchlaufener Addierer verstanden werden kann.

Die Auswertung ergibt, dass der vorgestellte Ansatz über das logarithmische Zahlenformat für vier bzw. sechs Iterationsschritte des CORDIC gleich viel oder weniger Signalverarbeitungsaufwand besitzt. Bezogen auf die Latenz ist dies mit acht Iterationsschritten gleichzusetzen. Zur Bewertung muss ausgehend von diesen Ergebnissen die Rechengenauigkeit mit einbezogen werden, was daher im nächsten Schritt untersucht wird.

Rechengenauigkeit Für die Auswertung der Rechengenauigkeit werden ALFA-basierte Approximationen für die nichtlinearen Funktionen der Konvertermodule mit $\text{mae}_{ALFA} = 0,01$ (LZD_1), $\text{mae}_{ALFA} = 0,001$ (LZD_2) und $\text{mae}_{ALFA} = 0,0001$ (LZD_3) betrachtet, die in jeweils 7, 68 und 681 Segmenten resultieren. Zudem wird ein Matlab-

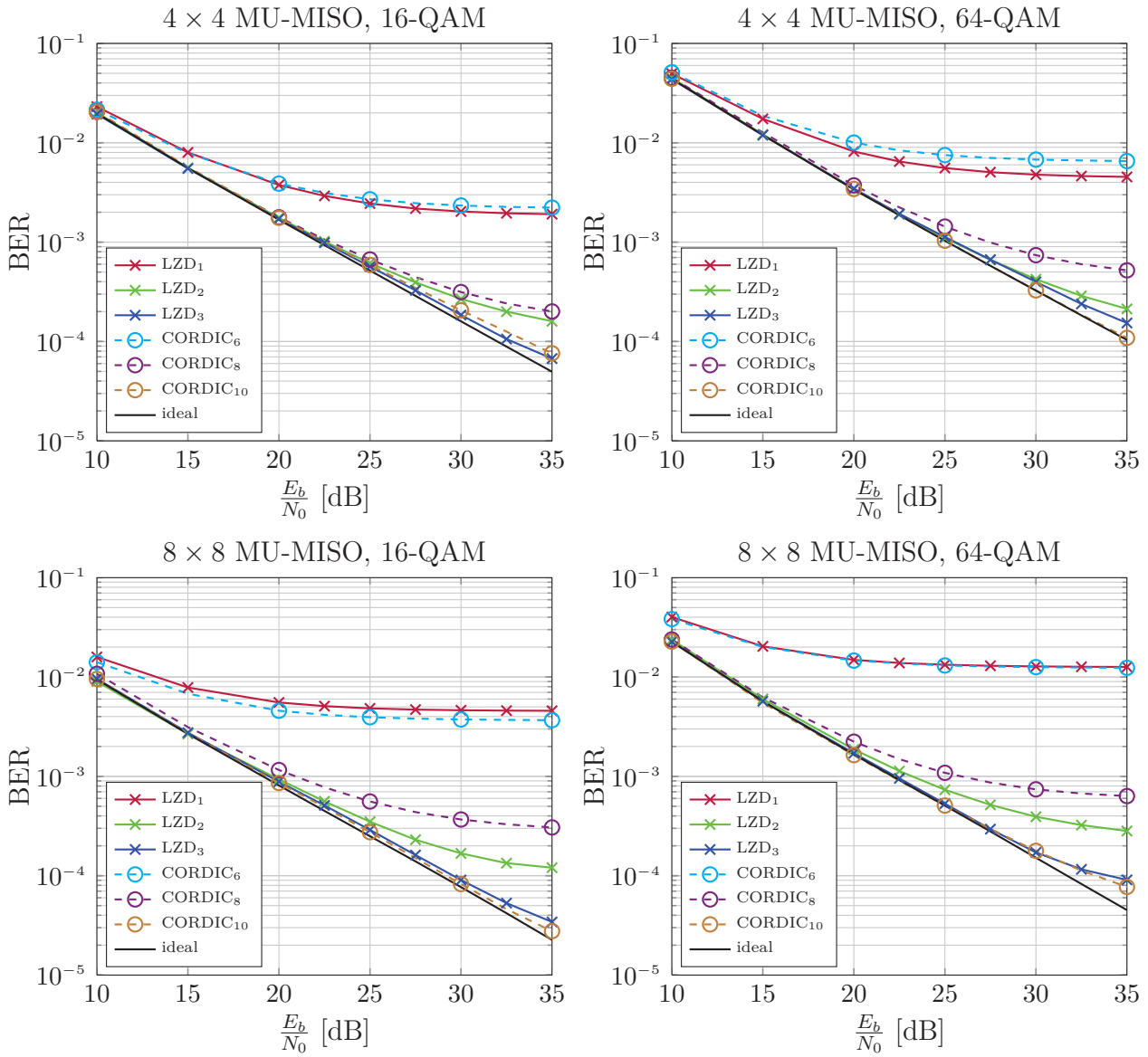


Abbildung 5.10: Bitfehlerkurven für ALFA-basierte QR-Zerlegungen zur Vorentzerrung mit drei unterschiedlichen Genauigkeiten sowie für drei CORDIC-Ansätze mit einer variierenden Anzahl an Iterationsschritten (m) und für eine ideale, gleitkommabasierte Matlab-Umsetzung. Betrachtet werden verschiedene Modulationen und eine unterschiedliche Anzahl an Antennen des in Kap. 5.1.3 angesprochenen MU-MISO-Systemmodells.

basiertes Referenzmodell erstellt. Für LZD_1 wird zudem eine Datenwortbreite von 16 Bit, für LZD_2 und LZD_3 von 18 Bit verwendet.

Die anwendungsspezifische Evaluation der Rechengenauigkeit wird für die in Kap. 5.3.2 beschriebene THP-ZF Vorentzerrung eines 4×4 bzw. 8×8 Mehrantennensys-

Tabelle 5.4: Performance-Ergebnisse der ASIC-Implementierungen von LZD_1 und LZD_3 für die 4 × 4 Antennenkonfiguration der QR-Zerlegung sowie ein Vergleich zu aktuellen Referenzen.

Referenz	Prozessgröße [nm]	Frequenz [MHz]	Komplexität [kGE]	Leistungsverbrauch [mW]	Durchsatz [MQRZ/s]
LZD_1	**130**	**145**	**17,93**	**5,68**	**0,51**
LZD_3	**130**	**129**	**22,38**	**5,73**	**0,45**
[113]	180	272	-	105	0,71
[47]	180	100	111	319	25
[72]	180	166	48,7	-	2,08
[86]	130	270	36	-	6,76

tems durchgeführt. Es werden zudem 16- und 64-QAM Konstellationen betrachtet. Für diese vier Anwendungsfälle wird eine sendeseitig bekannte Rayleigh-Kanalmatrix **H** verwendet. Zusätzlich wird AWGN-Rauschen empfangsseitig berücksichtigt.

Für die vier beschriebenen Szenarien werden BER-Kurven nach der Detektion für alle LZD-, sowie für CORDIC-Realisierungen mit unterschiedlicher Anzahl an Iterationsschritten erstellt. Die Auswertung ergibt, dass LZD_2 und LZD_3 für jede Konfiguration bessere Werte als eine CORDIC-Realisierung mit acht, teilweise sogar mit zehn Iterationsschritten erzielt. LZD_1 erreicht in nahezu allen Fällen die Rechengenauigkeit von $m = 6$ und überbietet somit hinsichtlich der Gesamtanzahl an Addierern den CORDIC-Ansatz. Die BER-Kurven sind in Abb. 5.10 dargestellt.

Aufgrund des niedrigen Quantisierungsfaktors ist eine hohe Anzahl an Segmenten für LZD_3 erkennbar, die mit erhöhtem Signalverarbeitungsaufwand zur Ansteuerung einhergeht. Es muss daher im nächsten Schritt überprüft werden, in welchem Ausmaß sich dies negativ auf die Komplexität auswirkt.

Implementierung Die CMOS-Implementierung der QR-Zerlegung basierend auf logarithmischer Zahlendarstellung wird für LZD_1 und LZD_3 durchgeführt, ausgelegt auf ein 4 × 4 Mehrantennensystem. Das Matlab-Modell dient hierbei als Spezifikations- und Verifikationsreferenz für eine entsprechende VHDL-Beschreibung. Die nichtlinearen Funktionen innerhalb der Konvertermodule werden mittels ALFA-Synthese erstellt. Durch Logik- und Layoutsynthese wird der Hardwareentwurf in ein Layout übersetzt. Als Technologie wird auf den 130nm Faraday Prozess der Firma UMC zurückgegriffen [119]. Tab. 5.4 weist den logarithmischen Ansatz zur Berechnung der QR-Zerlegung im Vergleich zu anderen Referenzen als performant hinsichtlich Komplexität und Leistungsverbrauch aus.

Diese Resultate zeigen, dass das logarithmische Zahlenformat einen durchaus effizienten Ansatz zur Berechnung der QR-Zerlegung darstellt. So werden, verglichen mit konventionellen CORDIC-Ansätzen, bei geringerer algorithmischer Komplexität gleichbleibende oder sogar bessere Ergebnisse bezogen auf die Übertragungsqualität erzielt. Zudem weist die gewählte Implementierung sehr gute Werte hinsichtlich Komplexität und Energieverbrauch auf. Der eher geringe Datendurchsatz kann durch Anwendung von Implementierungstechniken aus Kap. 2.3 wie Parallelisierung gesteigert werden.

5.4 Zusammenfassung

In diesem Kapitel wurden Hardwareimplementierungen ausgewählter Basisband-Signalverarbeitungsmodule des Mobilfunks behandelt. Sowohl für die CFO-Kompensation als auch die QR-Zerlegung sollten dabei nicht triviale, elementare Funktionen berechnet werden, was durch die in Kap. 3.7 vorgestellte ALFA-basierte Funktionsapproximation umgesetzt wurde. Der Fokus lag auf Matrixrotationen, für die in beiden Fällen entsprechende Sinus- und Kosinusterme zu bestimmen waren. Bei der CFO-Kompensation geschah dies anhand eines vorgegebenen Frequenzversatzes. So wurde, aufgrund der Abhängigkeit von nur einer Variablen, die Funktion direkt approximiert. Durch den Zusammenhang in Gl. (5.11) war zudem nur eine einzige Funktionsapproximation für beide trigonometrischen Funktionen notwendig. Bei der QR-Zerlegung wurden die Sinus- und Kosinusfunktionen über die in Gl. (5.26) und Gl. (5.27) gegebenen trigonometrischen Funktionen bestimmt. Durch Transformation der Werte in das logarithmische Zahlenformat ließen sich sämtliche benötigten Rechenschritte durch die arithmetischen Basisoperatoren Addition und Subtraktion realisieren. Bei diesem Ansatz wurden daher zwei entsprechende Konverterfunktionen zur Transformation von Zahlenwerten zwischen Festkomma- und logarithmischem Zahlenformat approximiert. Zur Evaluation der Ergebnisse wurden beide Entwürfe simuliert, implementiert und mit aktuellen Referenzen verglichen. In beiden Anwendungsfällen erzielten die vorgestellten ALFA-basierten Ansätze sehr gute Ergebnisse hinsichtlich Komplexität und Energieeffizienz. Für die CFO-Kompensation konnten zudem verhältnismäßig hohe Taktfrequenzen erreicht werden, weswegen die ALFA-Synthese als sehr geeigneter Ansatz zum Hardwareentwurf der Basisband-Signalverarbeitung im Mobilfunk angesehen werden kann.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

In dieser Arbeit wurden digitale Schaltungen für ausgewählte anwendungsspezifische Problemstellungen entworfen, wobei sowohl festverdrahtete als auch programmierbare Implementierungen verwendet wurden. Die zugrunde liegenden Algorithmen bestehen dabei aus nicht trivialen, elementaren Funktionen, für die in der Digitaltechnik kein einheitliches Verfahren zur Berechnung existiert. Kern dieser Arbeit war die näherungsweise Bestimmung einer Lösung besagter elementarer Funktionen anhand eines speziellen Synthesewerkzeugs, der automatisierten linearen Funktionsapproximation (ALFA), die sich als Kombination mehrerer bekannter Ansätze interpretieren lässt.

Das grundlegende Prinzip der ALFA-Synthese ist eine hardwareoptimierte näherungsweise Abbildung einer gegebenen Funktion. So wird der zugehörige Funktionsverlauf für einen vorgegebenen Abschnitt in mehrere Teilfunktionen unterteilt, die durch multipliziererfreie Geradengleichungen mit extrem geringer algorithmischer Latenz - bestenfalls wird nur ein Addierer benötigt - approximiert werden. Die zumeist nicht-uniformen Segmente werden dabei basierend auf hardwareeffizienter Bisektion durch eine vorgegebene Heuristik automatisch bestimmt. Für die resultierende Näherung können zudem die mittlere Rechengenauigkeit, der Quantisierungsfaktor der Geradengleichung sowie die Zulässigkeit von Sprungstellen der approximierten Funktion spezifiziert werden. Somit ermöglicht die ALFA-Synthese den zeiteffizienten Entwurf elementarer Funktionen, was eine experimentelle Analyse mehrerer resultierender HDL-Beschreibungen gleicher Funktion, aber unterschiedlicher Konfiguration ermöglicht.

Eine wichtige Anwendung für die ALFA-Synthese ist die Erzeugung von Konvertermodulen zur Transformation von Zahlenwerten zwischen Festkomma- und logarithmischem Zahlenformat. Neben Schiebeoperatoren und Multiplexern muss hierfür auch der Logarithmus bzw. die Zweierpotenz berechnet werden, was im Rahmen dieser Arbeit durch Funktionsapproximation realisiert wurde. Die Transformation in das logarithmische Zahlenformat erlaubt für einige elementare Funktionen, z.B. die Quadratwurzelberechnung, eine signifikante Reduktion des Signalverarbeitungsaufwands. Die ALFA-Synthese wurde daher nicht nur für die explizite Berechnung von elementaren

Funktionen im gegebenen Algorithmus, sondern auch zur Transformation der Zahlendarstellung verwendet.

Im Bereich drahtloser Sensornetze wurde die ALFA-Synthese zur Messwertvorhersage für den Transport verderblicher Waren im Container eingesetzt. Die Auswahl fiel durch allgemeinen Vergleich unterschiedlicher Ansätze auf den KNN-Prädiktor, der einerseits hohe Prädiktionsgenauigkeit erzielt, andererseits jedoch hohe Rechenzeit benötigt. Durch die ALFA-Synthese können kritische Terme, u.a. die Aktivierungsfunktionen hyperbolischer Tanges oder die Sigmoidfunktion, näherungsweise bestimmt werden. Um hierbei möglichst hohe Prädiktionsgenauigkeiten zu erzielen, musste eine hohe Rechengenauigkeit spezifiziert werden. Zudem wurden zur Vermeidung unerwünschter Effekte wie Aufschwingen die Funktionsapproximationen mit kontinuierlichem Verlauf erstellt. Der ALFA-basierte Ansatz wurde als ASIP umgesetzt. Als Referenzentwurf diente die Implementierung eines festverdrahteten CORDIC-Hardwarebeschleunigers. Der Vergleich der Ergebnisse wies den ALFA-ASIP als geeignete Lösung für sensorbasierte Berechnung des KNN-Algorithmus aus, da er prinzipiell als vollständige Signalverarbeitungseinheit eines Sensorknotens eingesetzt werden kann, der CORDIC-Hardwarebeschleuniger muss hingegen an einen Mikrocontroller angeschlossen werden.

Die ALFA-Synthese wurde des Weiteren zur näherungsweisen Umsetzung trigonometrischer Funktionen eingesetzt, die für ausgewählte Algorithmen aus dem Bereich Mobilfunk eingesetzt wurden. Hierzu gehört zum Einen der Entwurf digitaler Oszillatoren zur Frequenzoffsetkompensation, die zur Synchronisation empfangener Daten anhand eines vorgegebenen Frequenzversatzes verwendet werden. Eine anwendungsspezifische Analyse mehrerer ALFA-basierter Entwürfe zeigte, dass schon für geringe Rechengenauigkeiten akzeptable Ergebnisse entstehen. Somit wurde die ALFA-Synthese hinsichtlich hoher Performance für Komplexität und Taktfrequenz durchgeführt, was unter Anwendung des Zeitmultiplexverfahrens zu einer Gesamtanzahl von sechs Addieren bei einer Latenz von fünf Takten führte. Der erreichte Datendurchsatz ist für aktuelle Übertragungsstandards, z.B. LTE, ausreichend. Ein abschließender Vergleich mit aktuellen Referenzen zeigte, dass der ALFA-basierte Entwurf einen sehr performanten Ansatz darstellt.

Zum Anderen wurde die ALFA-Synthese für eine QR-Zerlegung unter Verwendung des logarithmischen Zahlensystems verwendet. Der Fokus lag dabei auf einer möglichst minimalen Umsetzung bezogen auf Komplexität. Durch Umformung konnte der Givens-Roation-basierte QR-Algorithmus dahingehend umgeformt werden, dass möglichst viele Operationen im logarithmischen Zahlenformat berechenbar sind. Die Berechnung arithmetischer Basisoperatoren wurde durch die oben angesprochenen Konvertermodule ermöglicht, die durch die ALFA-Synthese in verschiedenen Konfigurationen realisiert wurden. Die Bewertung erfolgte durch den Verglich der algorithmischen Latenz sowie der Bitfehlerkurven mit CORDIC-Referenzmodellen. In beiden Fällen zeigte der logarithmische Ansatz gleiche, zumeist bessere Performance auf. Auch der abschlie-

ßende Vergleich der Ergebnisse der Layoutsynthese mit aktuellen Referenzen wies den ALFA-basierten Ansatz als performant hinsichtlich Energieverbrauch und Komplexität aus.

Zusammenfassend kann der im Rahmen dieser Arbeit entwickelte und verwendete Ansatz zur automatischen linearen Funktionsapproximation als sehr leistungsstarkes Synthesewerkzeug zum zeiteffizienten Entwurf von Schaltungen im anwendungsspezifischen Mobilfunk angesehen werden. Anhand ausgewählter Algorithmen wurde gezeigt, dass für die Entwurfskriterien Energieeffizienz, Komplexität und Taktdauer, die je nach Anwendung unterschiedlich starke Bedeutung haben, eine hohe Performance erzielt werden kann. So waren die Ergebnisse zumeist deutlich besser als vergleichbare Referenzen, wodurch die Verwendung der ALFA-Synthese für zukünftigen anwendungsspezifischen Schaltungsentwurf sinnvoll erscheint.

6.2 Ausblick

Die vorgestellte automatische Funktionsapproximation hat i.d.R. in den im Rahmen der Arbeit behandelten Anwendungen zu hoher Performance geführt. Für die einzelnen Anwendungen lassen sich allerdings weitergehende Möglichkeiten erkennen, die zu Steigerungen der Performance oder des Bedienkomforts führen können. Für die KNN-basierte Prädiktion im Containerumfeld ist der Einsatz des ASIP als vollständige Signalverarbeitungseinheit des Sensorknotens nur theoretisch untersucht, eine praktische Umsetzung mit abschließender Evaluation steht noch aus. Hierfür muss zum Einen die HDL-Implementierung um die entsprechenden Peripheriemodule, z.B. USART, ergänzt werden. Zum Anderen muss der Hardwareentwurf dann als FPGA oder ASIC in eine entsprechende Sensorknotenumgebung eingebettet werden.

Für die QR-Zerlegung im Mobilfunk lassen sich zwei denkbare Weiterentwicklungen erkennen: So ist für die Detektion im Empfänger zumeist ein Hardwareentwurf mit hohem Datendurchsatz notwendig, um die Anforderungen moderner Mobilfunkstandards zu erfüllen. Hierfür ist eine entsprechende Implementierung des beschriebenen Ansatzes z.B. als systolisches Array notwendig. Zudem kann anstelle der Transformation zwischen den Zahlenformaten versucht werden, die arithmetischen Basisoperationen direkt in das logarithmische Zahlenformat abzubilden. Entsprechende allgemeine Ansätze wurden in der Vergangenheit bereits untersucht.

Neben den anwendungsspezifischen Erweiterungen kann auch die ALFA-Synthese selbst weiterentwickelt werden. Dabei sind verschiedene Vorgehensweisen denkbar, z.B. eine komplexere Heuristik zur Parameterextraktion, die unterschiedliche Verfahren zur Bestimmung der Segmente vergleicht und sich für das bestgeeignete Verfahren entscheidet.

A Automatische lineare Funktionsapproximation

A.1 Pseudocode ALFA-Funktionen

Alg. A.1 bzw. Alg. A.2 zeigt die Pseudocodebeschreibung zur Berechnung der quantisierten Steigung bzw. des Achsenabschnitts für die in Kap. 3.7.1 beschriebene Heuristik zur Parameterextraktion der ALFA-Synthese. Die vorliegenden Funktionen behandeln den Fall der nicht-stetigen Approximation.

Algorithmus A.1: Bestimmung der quantisierten Steigung

```
double estimateQuantizedGradient(double[] f_c, short qf) {
  double a_{1,c} = 0;
  // Bestimmung der Referenzsteigung mittels linearer Regression
  double a_{1,lin} = polyfit(f_c, 1);
  for short i=1 to qf
    a_{1,c} =+ arg min_{x̂} ((a_{1,lin} − a_{1,c}) ± 2^{x̂});
  return a_{1,c};
}
```

Algorithmus A.2: Bestimmung des Achsenabschnitts

```
double estimateBias(double a_{1,c}, double[] f_c, double[] x_c) {
  double a_{0,c} = 0;
    a_{0,c} = arg min_{â_{0,c}} (f_c − (a_{1,c} x_c + â_{0,c}));
  return a_{0,c};
}
```

A.2 Beispiel Parameterextraktion

Tab. A.1 zeigt das Ergebnis der ALFA-Parameterextraktion der in Kap. 3.7 verwendeten Funktion $f(x) = \frac{3}{2}x^5 - 7x^3 + \frac{11}{2}x + 2$ mit dem Quantisierungsfaktor $QF = 1$ und dem mittleren absoluten Fehler $\text{mae}_{ALFA} = 0,05$. Anhand dieser Daten kann im nachfolgenden HDL-Erzeugungsschritt eine entsprechende Verilog-Beschreibung erstellt werden.

Tabelle A.1: ALFA-Parameter der in Gl. (3.61) und Gl. (3.62) gegebenen Approximations- und Segmentgleichung für $f(x) = \frac{3}{2}x^5 - 7x^3 + \frac{11}{2}x + 2$ mit $QF = 1$ und $\text{mae}_{ALFA} = 0,05$ (siehe Abb. 3.9b). Die erste Zeile ist für den Startpunkt A von $f(x)$ vorgesehen.

i	$a_{i,0}$	$a_{i,1}$	seg(i)	seg(i)−seg($i-1$)	sign($\lambda_{i,0}$)	$\lambda_{i,0}$
	0.0000000000e+00	0.0000000000e+00	0.0000000000e+00	0.0000000000e+00	0.0000000000e+00	0.0000000000e+00
0	2.0920104980e+00	4.0000000000e+00	1.2500000000e-01	1.2500000000e-01	1.0000000000e+00	2.0000000000e+00
1	2.2354431152e+00	4.0000000000e+00	2.5000000000e-01	1.2500000000e-01	1.0000000000e+00	2.0000000000e+00
2	2.9408874512e+00	2.0000000000e+00	5.0000000000e-01	2.5000000000e-01	1.0000000000e+00	1.0000000000e+00
3	4.4684448242e+00	-1.0000000000e+00	7.5000000000e-01	2.5000000000e-01	-1.0000000000e+00	0.0000000000e+00
4	6.4952697754e+00	-4.0000000000e+00	8.7500000000e-01	1.2500000000e-01	-1.0000000000e+00	2.0000000000e+00
5	6.3162536621e+00	-4.0000000000e+00	9.3750000000e-01	6.2500000000e-02	-1.0000000000e+00	2.0000000000e+00
6	6.1735839844e+00	-4.0000000000e+00	9.6875000000e-01	3.1250000000e-02	-1.0000000000e+00	2.0000000000e+00
7	6.0610961914e+00	-4.0000000000e+00	1.0000000000e+00	3.1250000000e-02	-1.0000000000e+00	2.0000000000e+00
8	9.9786376953e+00	-8.0000000000e+00	1.1250000000e+00	1.2500000000e-01	-1.0000000000e+00	3.0000000000e+00
9	9.8514404297e+00	-8.0000000000e+00	1.2500000000e+00	1.2500000000e-01	-1.0000000000e+00	3.0000000000e+00
10	9.7356567383e+00	-8.0000000000e+00	1.3750000000e+00	1.2500000000e-01	-1.0000000000e+00	3.0000000000e+00
11	4.1420593262e+00	-4.0000000000e+00	1.4375000000e+00	6.2500000000e-02	-1.0000000000e+00	2.0000000000e+00
12	4.0266723633e+00	-4.0000000000e+00	1.5000000000e+00	6.2500000000e-02	-1.0000000000e+00	2.0000000000e+00
13	-1.3364868164e+00	-5.0000000000e-01	1.6250000000e+00	1.2500000000e-01	-1.0000000000e+00	-1.0000000000e+00
14	-1.5307281494e+01	8.0000000000e+00	1.7500000000e+00	1.2500000000e-01	1.0000000000e+00	3.0000000000e+00
15	-2.9366851807e+01	1.6000000000e+01	1.8125000000e+00	6.2500000000e-02	1.0000000000e+00	4.0000000000e+00
16	-2.9334899902e+01	1.6000000000e+01	1.8437500000e+00	3.1250000000e-02	1.0000000000e+00	4.0000000000e+00
17	-2.9232788086e+01	1.6000000000e+01	1.8593750000e+00	1.5625000000e-02	1.0000000000e+00	4.0000000000e+00
18	-2.9130584717e+01	1.6000000000e+01	1.8750000000e+00	1.5625000000e-02	1.0000000000e+00	4.0000000000e+00
19	-5.9173156738e+01	3.2000000000e+01	1.9062500000e+00	3.1250000000e-02	1.0000000000e+00	5.0000000000e+00
20	-5.9290802002e+01	3.2000000000e+01	1.9375000000e+00	3.1250000000e-02	1.0000000000e+00	5.0000000000e+00
21	-5.9279357910e+01	3.2000000000e+01	1.9687500000e+00	3.1250000000e-02	1.0000000000e+00	5.0000000000e+00
22	-5.9180633545e+01	3.2000000000e+01	1.9843750000e+00	1.5625000000e-02	1.0000000000e+00	5.0000000000e+00
23	-5.9069488525e+01	3.2000000000e+01	1.9999694824e+00	1.5625000000e-02	1.0000000000e+00	5.0000000000e+00

A.3 Polynomiale Approximation Sigmoidfunktion

A.3.1 *Forward prediction*

Abb. A.1 zeigt einen graphischen Überblick über die polynomial approximierte Sigmoidfunktion $\tilde{\phi}_{PA,n}(x)$, mit n als Polynomgrad, im Vergleich zum Verlauf der Originalfunktion $\tilde{\phi}(x)$. Zudem ist der mittlere absolute Fehler im Bereich $-1 \leq x < 1$ angegeben.

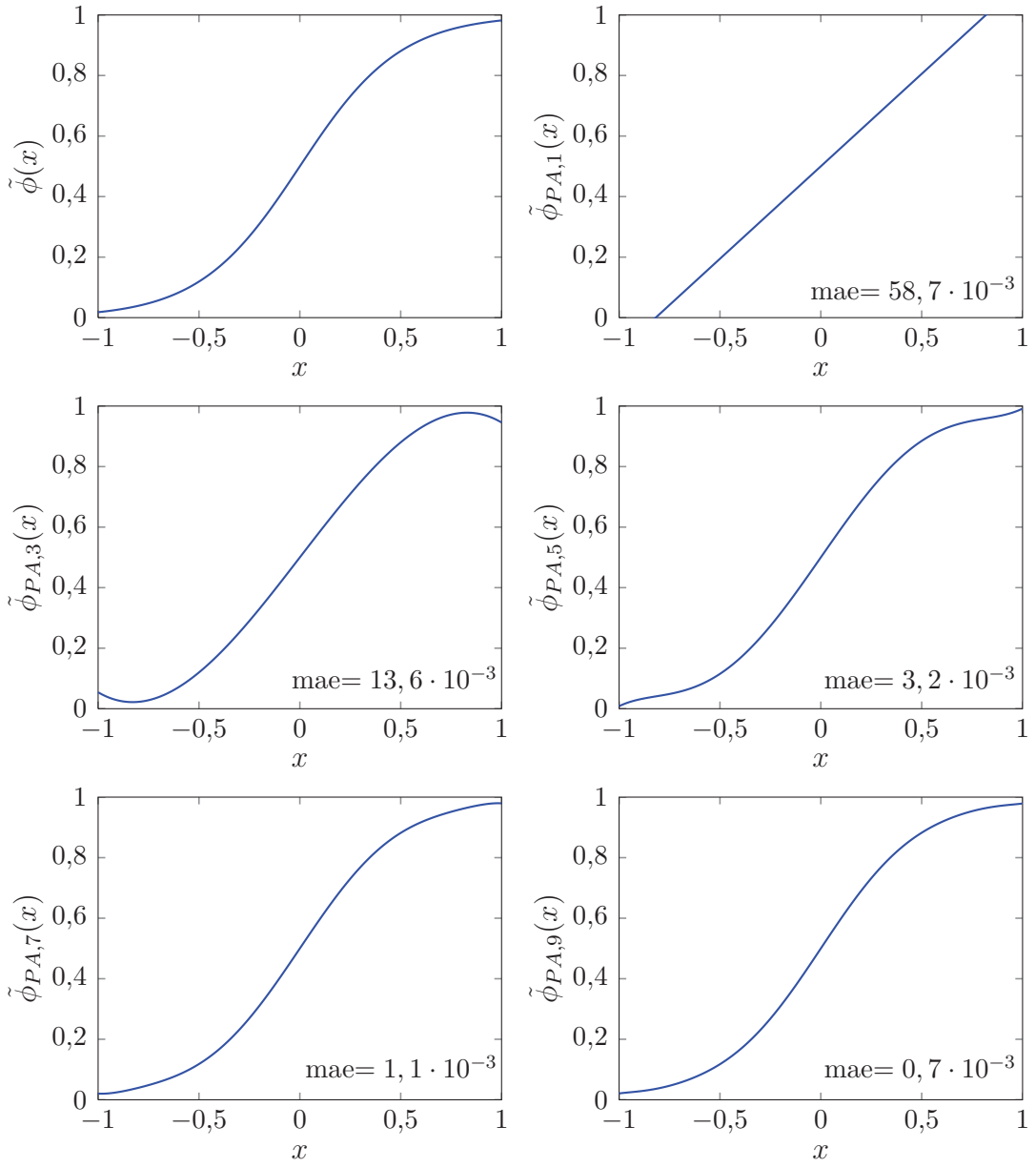


Abbildung A.1: Funktionsverläufe der in Kap. 4.3.1 mittels polynomialer Regression approximierten Sigmoidfunktionen und der Originalfunktion.

A.3.2 *Backpropagation*

Abb. A.2 zeigt einen graphischen Überblick über die polynomial approximierten Gradienten der Sigmoidfunktion $\tilde{\phi}'_{PA,n}(x) = \tilde{\phi}_{PA,n}(x)(1 - \tilde{\phi}_{PA,n}(x))$, mit n als Polynomgrad, im Vergleich zum Verlauf des Gradienten der Originalfunktion $\tilde{\phi}'(x)$. Zudem ist der mittlere absolute Fehler im Bereich $-1 \leq x < 1$ angegeben.

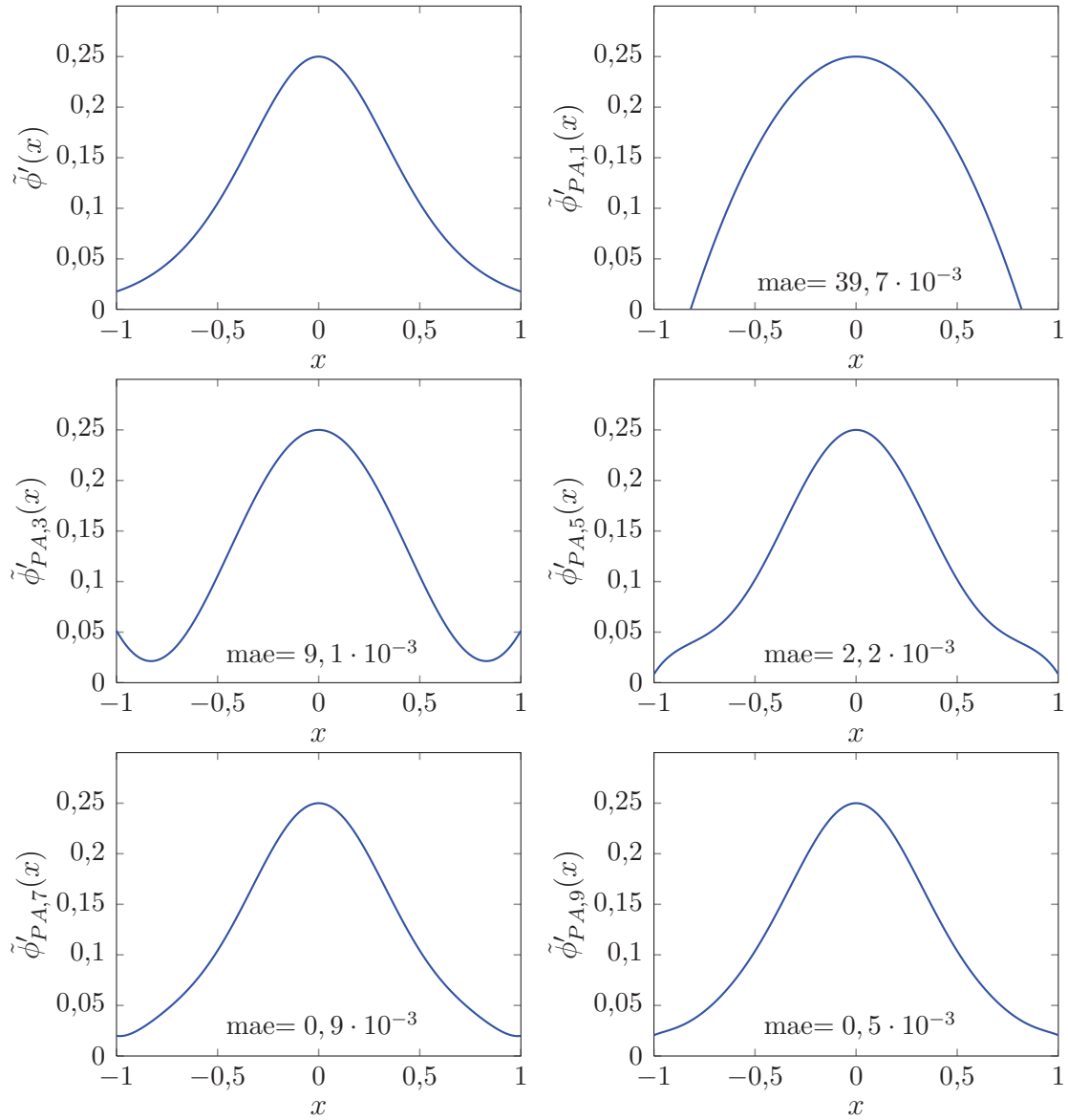


Abbildung A.2: Funktionsverläufe der in Kap. 4.3.1 mittels polynomialer Regression approximierten Gradienten der Sigmoidfunktion und der Originalfunktion.

.

A.4 Uniforme Segmentierung der Sigmoidfunktion

A.4.1 *Forward prediction*

Abb. A.3 zeigt einen graphischen Überblick über die uniform approximierten Sigmoidfunktionen $\tilde{\phi}^i_{SA}(x)$, mit i als Anzahl an Segmenten, im Vergleich zum Verlauf der Originalfunktion $\tilde{\phi}(x)$. Zudem ist der mittlere absolute Fehler im Bereich $-1 \leq x < 1$ angegeben.

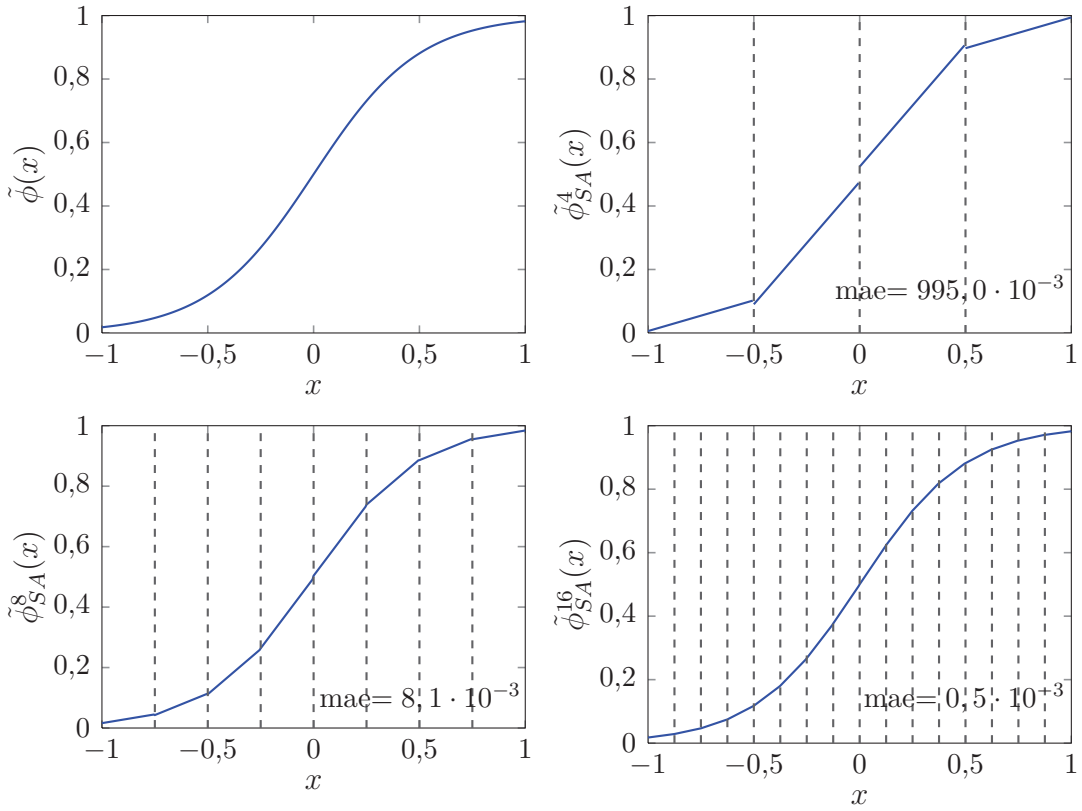


Abbildung A.3: Funktionsverläufe der in Kap. 4.3.2 mittels uniformer Segmentierung approximierten Sigmoidfunktionen und der Originalfunktion.

A.4.2 *Backpropagation*

Abb. A.4 zeigt einen graphischen Überblick über die uniform approximierten Gradienten der Sigmoidfunktion $\tilde{\phi}'^{n}_{SA}(x) = \tilde{\phi}^{n}_{SA}(x)(1-\tilde{\phi}^{n}_{SA}(x))$, mit i als Anzahl an Segmenten, im Vergleich zum Verlauf des Gradienten der Originalfunktion $\tilde{\phi}'(x)$. Zudem ist der mittlere absolute Fehler im Bereich $-1 \leq x < 1$ angegeben.

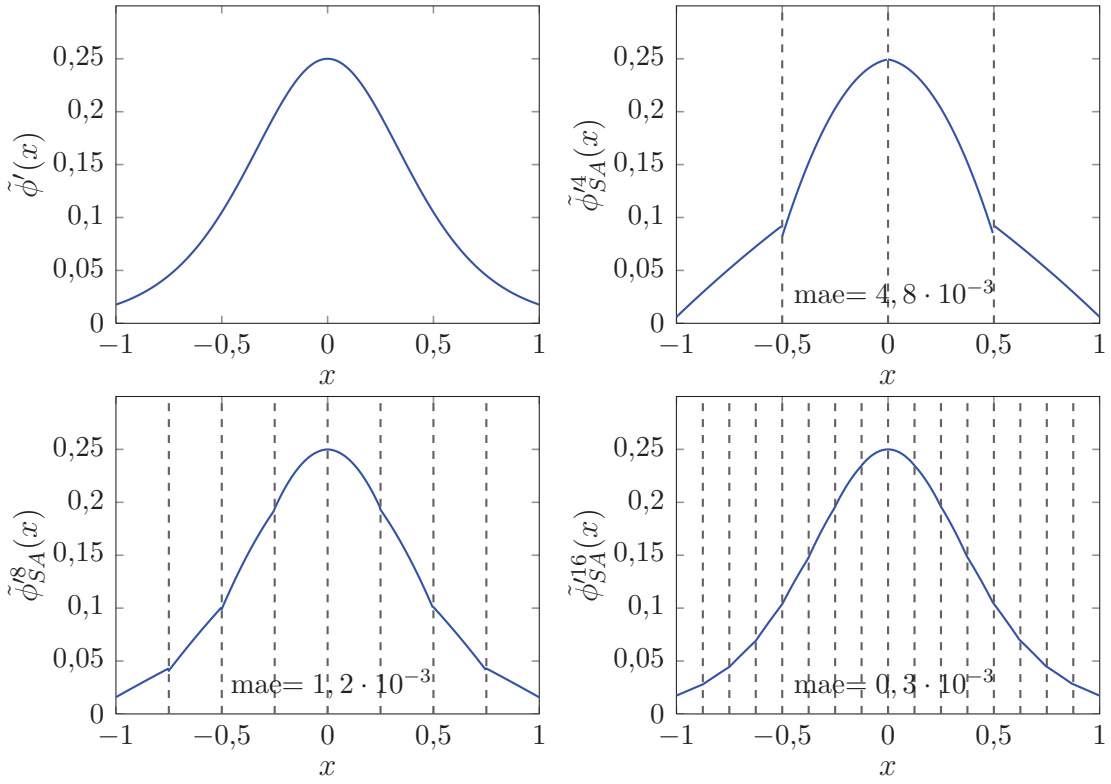


Abbildung A.4: Funktionsverläufe der in Kap. 4.3.2 mittels uniformer Segmentierung approximierten Sigmoidfunktionen und der Originalfunktion.

A.5 ALFA-ASIP

A.5.1 Instruktionssatz

Vollständiger Instruktionssatz des in Kap. 4.4.2 beschriebenen ASIP zur Berechnung MLP-basierter KNN-Strukturen, der aus 16 Bit, 32 Bit und Sprungbefehlen besteht.

16 Bit Befehle

Mnemonik	Operanden	Operation	Adressmodus
add	$\mathbf{Rx}_{S/D}$, const	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D + \mathbf{Rx}_S$/const	;as==0 && ad==0
		$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D +$ M[$\mathbf{Rx}_S$]	;as==1 && ad==0
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] + $\mathbf{Rx}_S$/const	;as==0 && ad==1
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] + M[$\mathbf{Rx}_S$]	;as==1 && ad==1
add_acc	$\mathbf{Rx}_{S/D}$	ACCU $\leftarrow \mathbf{Rx}_D + \mathbf{Rx}_S$	;ad==0
		ACCU $\leftarrow$ M[$\mathbf{Rx}_D$] + $\mathbf{Rx}_S$	;ad==1
addc	$\mathbf{Rx}_{S/D}$, const	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D + \mathbf{Rx}_S$/const + C	;as==0 && ad==0
		$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D +$ M[$\mathbf{Rx}_S$] + C	;as==1 && ad==0
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] + $\mathbf{Rx}_S$/const + C	;as==0 && ad==1
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] + M[$\mathbf{Rx}_S$] + C	;as==1 && ad==1
addc_acc	$\mathbf{Rx}_{S/D}$	ACCU $\leftarrow \mathbf{Rx}_D + \mathbf{Rx}_S + C$	;ad==0
		ACCU $\leftarrow$ M[$\mathbf{Rx}_D$] + $\mathbf{Rx}_S + C$	;ad==1
sub	$\mathbf{Rx}_{S/D}$, const	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D - \mathbf{Rx}_S$/const	;as==0 && ad==0
		$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D -$ M[$\mathbf{Rx}_S$]	;as==1 && ad==0
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $- \mathbf{Rx}_S$/const	;as==0 && ad==1
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $-$ M[$\mathbf{Rx}_S$]	;as==1 && ad==1
sub_acc	$\mathbf{Rx}_{S/D}$	ACCU $\leftarrow \mathbf{Rx}_D - \mathbf{Rx}_S$	;ad==0
		ACCU $\leftarrow$ M[$\mathbf{Rx}_D$] $- \mathbf{Rx}_S$	;ad==1
subc	$\mathbf{Rx}_{S/D}$, const	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D - \mathbf{Rx}_S$/const + C	;as==0 && ad==0
		$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D -$ M[$\mathbf{Rx}_S$] + C	;as==1 && ad==0
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $- \mathbf{Rx}_S$/const + C	;as==0 && ad==1
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $-$ M[$\mathbf{Rx}_S$] + C	;as==1 && ad==1
subc_acc	$\mathbf{Rx}_{S/D}$	ACCU $\leftarrow \mathbf{Rx}_D - \mathbf{Rx}_S + C$	;ad==0
		ACCU $\leftarrow$ M[$\mathbf{Rx}_D$] $- \mathbf{Rx}_S + C$	;ad==1
and	$\mathbf{Rx}_{S/D}$, const	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \wedge \mathbf{Rx}_S$/const	;as==0 && ad==0
		$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \wedge$ M[$\mathbf{Rx}_S$]	;as==1 && ad==0
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\wedge \mathbf{Rx}_S$/const	;as==0 && ad==1
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\wedge$ M[$\mathbf{Rx}_S$]	;as==1 && ad==1
and_acc	$\mathbf{Rx}_{S/D}$	ACCU $\leftarrow \mathbf{Rx}_D \wedge \mathbf{Rx}_S$	;ad==0
		ACCU $\leftarrow$ M[$\mathbf{Rx}_D$] $\wedge \mathbf{Rx}_S$	;ad==1
mov	$\mathbf{Rx}_{S/D}$, const	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_S$/const	;as==0 && ad==0
		$\mathbf{Rx}_D \leftarrow$ M[$\mathbf{Rx}_S$]	;as==1 && ad==0
		M[$\mathbf{Rx}_D$] $\leftarrow \mathbf{Rx}_S$/const	;as==0 && ad==1
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_S$]	;as==1 && ad==1

Mnemonik	Operanden	Operation	Adressmodus
mov_acc[1]		**mov** ACCU, $\mathbf{Rx}_S$/@AI$\mathbf{Rx}_S$	
or	$\mathbf{Rx}_{S/D}$,	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \vee \mathbf{Rx}_S$/const	;as==0 && ad==0
	const	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \vee$ M[$\mathbf{Rx}_S$]	;as==1 && ad==0
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\vee$ $\mathbf{Rx}_S$/const	;as==0 && ad==1
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\vee$ M[$\mathbf{Rx}_S$]	;as==1 && ad==1
or_acc	$\mathbf{Rx}_{S/D}$	ACCU $\leftarrow \mathbf{Rx}_D \vee \mathbf{Rx}_S$	;ad==0
		ACCU $\leftarrow$ M[$\mathbf{Rx}_D$] $\vee \mathbf{Rx}_S$	;ad==1
xor	$\mathbf{Rx}_{S/D}$,	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \oplus \mathbf{Rx}_S$/const	;as==0 && ad==0
	const	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \oplus$ M[$\mathbf{Rx}_S$]	;as==1 && ad==0
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\oplus$ $\mathbf{Rx}_S$/const	;as==0 && ad==1
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\oplus$ M[$\mathbf{Rx}_S$]	;as==1 && ad==1
xor_acc	$\mathbf{Rx}_{S/D}$	ACCU $\leftarrow \mathbf{Rx}_D \oplus \mathbf{Rx}_S$	;ad==0
		ACCU $\leftarrow$ M[$\mathbf{Rx}_D$] $\oplus \mathbf{Rx}_S$	;ad==1
shl	$\mathbf{Rx}_{S/D}$,	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \ll \mathbf{Rx}_S$/const	;as==0 && ad==0
	const	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \ll$ M[$\mathbf{Rx}_S$]	;as==1 && ad==0
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\ll$ $\mathbf{Rx}_S$/const	;as==0 && ad==1
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\ll$ M[$\mathbf{Rx}_S$]	;as==1 && ad==1
shl_acc	$\mathbf{Rx}_{S/D}$	ACCU $\leftarrow \mathbf{Rx}_D \ll \mathbf{Rx}_S$	;ad==0
		ACCU $\leftarrow$ M[$\mathbf{Rx}_D$] $\ll \mathbf{Rx}_S$	;ad==1
shr	$\mathbf{Rx}_{S/D}$,	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \gg \mathbf{Rx}_S$/const	;as==0 && ad==0
	const	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \gg$ M[$\mathbf{Rx}_S$]	;as==1 && ad==0
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\gg$ $\mathbf{Rx}_S$/const	;as==0 && ad==1
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\oplus$ M[$\mathbf{Rx}_S$]	;as==1 && ad==1
shr_acc	$\mathbf{Rx}_{S/D}$	ACCU $\leftarrow \mathbf{Rx}_D \gg \mathbf{Rx}_S$	;ad==0
		ACCU $\leftarrow$ M[$\mathbf{Rx}_D$] $\gg \mathbf{Rx}_S$	;ad==1
mul	$\mathbf{Rx}_{S/D}$,	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \cdot \mathbf{Rx}_S$/const	;as==0 && ad==0
	const	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \cdot$ M[$\mathbf{Rx}_S$]	;as==1 && ad==0
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\cdot$ $\mathbf{Rx}_S$/const	;as==0 && ad==1
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\cdot$ M[$\mathbf{Rx}_S$]	;as==1 && ad==1
mul_acc	$\mathbf{Rx}_{S/D}$	ACCU $\leftarrow \mathbf{Rx}_D \cdot \mathbf{Rx}_S$	;ad==0)
		ACCU $\leftarrow$ M[$\mathbf{Rx}_D$] $\cdot \mathbf{Rx}_S$	;ad==1)
div	$\mathbf{Rx}_{S/D}$,	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \div \mathbf{Rx}_S$/const	;as==0 && ad==0
	const	$\mathbf{Rx}_D \leftarrow \mathbf{Rx}_D \div$ M[$\mathbf{Rx}_S$]	;as==1 && ad==0
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\div$ $\mathbf{Rx}_S$/const	;as==0 && ad==1
		M[$\mathbf{Rx}_D$] $\leftarrow$ M[$\mathbf{Rx}_D$] $\div$ M[$\mathbf{Rx}_S$]	;as==1 && ad==1
div_acc	**Rx**	ACCU $\leftarrow \mathbf{Rx}_D \div \mathbf{Rx}_S$	;ad==0)
		ACCU $\leftarrow$ M[$\mathbf{Rx}_D$] $\div \mathbf{Rx}_S$	;ad==1)

Mnemonik	Operanden	Operation	Adressmodus
smf	$\mathbf{Rx}_{S/D}$,	$\mathbf{Rx}_D \leftarrow \phi_{sigmoid}(\mathbf{Rx}_S/\texttt{const})$	;as==0 && ad==0
	const	$\mathbf{Rx}_D \leftarrow \phi_{sigmoid}(\mathbf{Rx}_S)$	;as==1 && ad==0
		$\texttt{M[}\mathbf{Rx}_D\texttt{]} \leftarrow \phi_{sigmoid}(\mathbf{Rx}_S/\texttt{const})$	;as==0 && ad==1
		$\texttt{M[}\mathbf{Rx}_D\texttt{]} \leftarrow \phi_{sigmoid}(\texttt{M[}\mathbf{Rx}_S\texttt{]})$	;as==1 && ad==1
smf_acc[1]		**smf** ACCU, $\mathbf{Rx}_S$/@AIR$\mathbf{x}_S$	
tanh	$\mathbf{Rx}_{S/D}$,	$\mathbf{Rx}_D \leftarrow \phi_{tanh}(\mathbf{Rx}_S/\texttt{const})$	;as==0 && ad==0
	const	$\mathbf{Rx}_D \leftarrow \phi_{tanh}(\texttt{M[}\mathbf{Rx}_S\texttt{]})$	;as==1 && ad==0
		$\texttt{M[}\mathbf{Rx}_D\texttt{]} \leftarrow \phi_{tanh}(\mathbf{Rx}_S/\texttt{const})$	;as==0 && ad==1
		$\texttt{M[}\mathbf{Rx}_D\texttt{]} \leftarrow \phi_{tanh}(\texttt{M[}\mathbf{Rx}_S\texttt{]})$	;as==1 && ad==1
tanh_acc[1]		**tanh** ACCU, $\mathbf{Rx}_S$/ @AIR$\mathbf{x}_S$	

32 Bit Befehle

Mnemonik	Operanden	Operation	Adressmodus
add	**Rx**,	**Rx** $\leftarrow$ **Rx** + #immediate	;ad==0
	#immediate	M[**Rx**] $\leftarrow$ M[$\mathbf{Rx}_D$] + #immediate	;ad==1
addc	**Rx**,	**Rx** $\leftarrow$ **Rx** + #immediate + C	;ad==0
	#immediate	M[**Rx**] $\leftarrow$ M[$\mathbf{Rx}_D$] + #immediate + C	;ad==1
sub	**Rx**,	**Rx** $\leftarrow$ **Rx** $-$ #immediate	;ad==0
	#immediate	M[**Rx**] $\leftarrow$ M[$\mathbf{Rx}_D$] $-$ #immediate	;ad==1
subc	**Rx**,	**Rx** $\leftarrow$ **Rx** $-$ #immediate + C	;ad==0
	#immediate	M[**Rx**] $\leftarrow$ M[$\mathbf{Rx}_D$] $-$ #immediate + C	;ad==1
and	**Rx**,	**Rx** $\leftarrow$ **Rx** $\wedge$ #immediate	;ad==0
	#immediate	M[**Rx**] $\leftarrow$ M[$\mathbf{Rx}_D$] $\wedge$ #immediate	;ad==1
mov	**Rx**,	**Rx** $\leftarrow$ #immediate	;ad==0
	#immediate	M[**Rx**] $\leftarrow$ #immediate	;ad==1
or	**Rx**,	**Rx** $\leftarrow$ **Rx** $\vee$ #immediate	;ad==0
	#immediate	M[**Rx**] $\leftarrow$ M[$\mathbf{Rx}_D$] $\vee$ #immediate	;ad==1
xor	**Rx**,	**Rx** $\leftarrow$ **Rx** $\oplus$ #immediate	;ad==0
	#immediate	M[**Rx**] $\leftarrow$ M[$\mathbf{Rx}_D$] $\oplus$ #immediate	;ad==1
shl	**Rx**,	**Rx** $\leftarrow$ **Rx** $\ll$ #immediate	;ad==0
	#immediate	M[**Rx**] $\leftarrow$ M[$\mathbf{Rx}_D$] $\ll$ #immediate	;ad==1
shr	**Rx**,	**Rx** $\leftarrow$ **Rx** $\gg$ #immediate	;ad==0
	#immediate	M[**Rx**] $\leftarrow$ M[$\mathbf{Rx}_D$] $\gg$ #immediate	;ad==1
mul	**Rx**,	**Rx** $\leftarrow$ smf(#immediate)	;ad==0
	#immediate	M[**Rx**] $\leftarrow$ M[$\mathbf{Rx}_D$] $\vee$ #immediate	;ad==1
div	**Rx**,	**Rx** $\leftarrow$ **Rx** $\oplus$ #immediate	;ad==0
	#immediate	M[**Rx**] $\leftarrow$ M[$\mathbf{Rx}_D$] $\oplus$ #immediate	;ad==1
smf	**Rx**,	**Rx** $\leftarrow \phi_{sigmoid}$(#immediate)	;ad==0
	#immediate	M[**Rx**] $\leftarrow \phi_{sigmoid}$(#immediate)	;ad==1
tanh	**Rx**,	**Rx** $\leftarrow \phi_{tanh}$(#immediate)	;ad==0
	#immediate	M[**Rx**] $\leftarrow \phi_{tanh}$(#immediate)	;ad==1

Sprungbefehle

Mnemonik	Operanden	Operation	Status
bc	`#offset`	**PC** ← `PM[#offset]`	
bcz	**R**x, `#offset`	**PC** ← `PM[#offset]`	;**R**x==0
bcp	**R**x, `#offset`	**PC** ← `PM[#offset]`	;**R**x>0
bcn	**R**x, `#offset`	**PC** ← `PM[#offset]`	;**R**x<0

Anmerkungen:

`as`	Indirekte Adressierung des Quelloperanden (Präfix @)
`ad`	Indirekte Adressierung des Zieloperanden (Präfix @)
C	Überlaufbit
1	Befehlsemulation: Verwendung des `ACCU`-Registers als Zieloperand
`const`	CPU-interner, konstanter Quelloperand: `ZERO, ONE, TWO, FOUR`
`#immediate`	Instruktionsbasierter, konstanter Quelloperand
`#offset`	Zieladresse bei Programmsprüngen
`M[x]`	Datenwort im Speicher an der Stelle `x`
`PM[x]`	Instruktion im Programmspeicher an der Stelle `x`
S	Kennzeichnung eines Operanden als Quelloperand
D	Kennzeichnung eines Operanden als Zieloperand

A.5.2 CPU-Register

Die nachfolgende Tabelle bietet einen Überblick über die CPU-Register des in Kap. 4.4.2 beschriebenen ALFA-ASIP sowie über die zugehörigen Eigenschaften.

Adresse	Name	Sonderfunktion	Anmerkung
000h	PC	–	*Program Counter*
001h	GCTRL	ZERO /TWO	Steuerregister zur Konfiguration des ASIP und zum Auslesen von Statusflags. Der indirekte bzw. Dreioperand-Zugriff für den Quelloperanden setzt diesen Operanden auf die Konstante Null bzw. Zwei.
002h	ACCU	ONE / FOUR	Zielregister für den Dreioperand-Zugriff. Der indirekte bzw. Dreioperand-Zugriff für den Quelloperanden setzt diesen Operanden auf die Konstante Eins bzw. Vier.
003h	R3	UOVLR	Bei **mul** bzw. **div** Operationen werden Unter- bzw. Überläufe in diesem Register gespeichert.
004h	R4	AIR0	Bei indirektem Zugriff wird die Adresse um den Wert in +AIR0− vorab bzw. nachfolgend erhöht bzw. erniedrigt.
005h	R5	AIR1	Bei indirektem Zugriff wird die Adresse um den Wert in +AIR1− vorab bzw. nachfolgend erhöht bzw. erniedrigt.
006h	R6	AIR2	Bei indirektem Zugriff wird die Adresse um den Wert in +AIR2− vorab bzw. nachfolgend erhöht bzw. erniedrigt.
007h	R7	AIR3	Bei indirektem Zugriff wird die Adresse um den Wert in +AIR3− vorab bzw. nachfolgend erhöht bzw. erniedrigt.
008h	R8	+AIR0−	Adresseninkrement bzw. -dekrement für AIR0 bei indirektem Zugriff.
009h	R9	+AIR1−	Adresseninkrement bzw. -dekrement für AIR1 bei indirektem Zugriff.
00Ah	R10	+AIR2−	Adresseninkrement bzw. -dekrement für AIR2 bei indirektem Zugriff.
00Bh	R11	+AIR3−	Adresseninkrement bzw. -dekrement für AIR3 bei indirektem Zugriff.
00Ch	R12	–	Datenregister
00Dh	R13	–	Datenregister
00Eh	R14	–	Datenregister
00Fh	R15	–	Datenregister

A.5.3 Befehlsgruppen

Abb. A.5 zeigt den bitgenauen Aufbau der verschiedenen Befehlsgruppen des in Kap. 4.4.2 beschriebenen ALFA-ASIP.

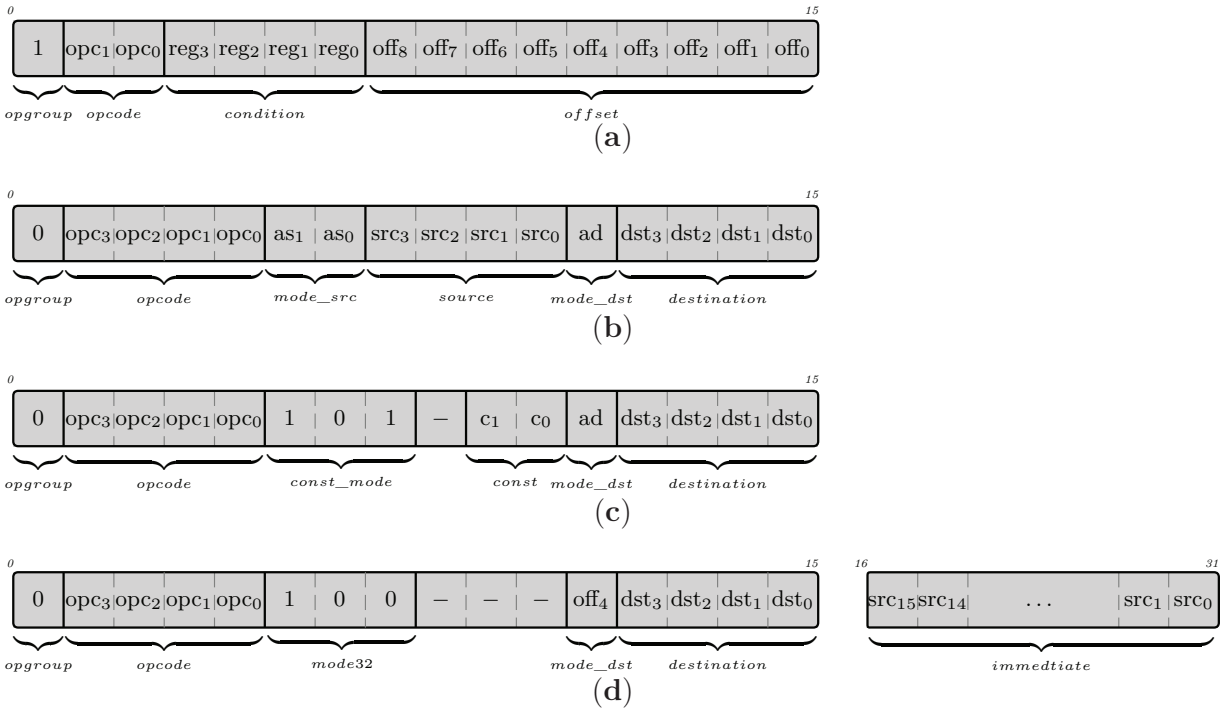


Abbildung A.5: Aufbau der Instruktionen des ALFA-ASIP mit **(a)** Sprungbefehlen, **(b)** 16 Bit Befehlen ohne Konstanten, **(c)** 16 Bit Befehlen mit Konstanten und **(d)** 32 Bit Befehlen.

Anmerkungen:

opgroup	Festlegung der Befehlsgruppe: 16 Bit, 32 Bit oder Sprungbefehle)
opcode	Festlegung des Befehls, z.B. mov, add, bcp
condition	CPU-Register für die Auswertung der Sprungbedingung
$offset$	Sprungadresse
mode_src	Adressmodus des Quelloperanden
source	Quellregister
mode_dst	Adressmodus des Zieloperanden
destination	Zielregister
const_mode	Festlegung des Konstantenmodus
const	Auswahl der Konstante: `ZERO`, `ONE`, `TWO`, `FOUR`
32*b_mode*	Festlegung der Befehlslänge auf 32 Bit
immediate	Im Befehl vorgegebene Konstante

A.5.4 Adressraum

Die nachfolgende Tabelle gibt einen Überblick über den Adressraum des in Kap. 4.4.2 beschriebenen ALFA-ASIP.

Adressbereich	Modul	Anmerkung
`000h - 00Fh`	CPU-Register	Register zum Zwischenspeichern und für zusätzliche spezielle Funktionen (siehe Anhang A.5.2)
`010h - 01Fh`	IO-Ports	IO-Port Ressourcen zur Konfiguration und Ansteuerung der parallelen Ausgangsports
`020h - 0FFh`	*reserved*	Reserviert für weitere Peripheriemodule
`100h - 17Fh`	`PM`	Programmspeicher
`800h - 8FFh`	`M`	Datenspeicher

A.5.5 Adressmodi

Die nachfolgende Tabelle zeigt einen Überblick über die verschiedenen Adressmodi des in Kap. 4.4.2 beschriebenen ALFA-ASIP.

as_1	as_0	ad	src_3/off_8	Kommentar
0	0	0	x	Direkter Zugriff auf die Werte der Quell- und Zielregister.
0	0	1	x	Direkter Zugriff auf den Wert des Quellregisters, das Zielregister fungiert als Zeiger (Adresse) auf den Speicher.
0	1	0	x	Das Quellregister fungiert als Zeiger (Adresse) auf den Speicher, direkter Zugriff auf den Wert des Zielregisters.
0	1	1	x	Quell- und Zielregister fungieren als Zeiger (Adresse) auf den Speicher.
1	0	0	0	Direkte Angabe des Quelloperanden im Befehl, direkter Zugriff auf die Werte des Zielregisters.
1	0	0	1	Quelloperand ist eine Konstante, direkter Zugriff auf den Wert des Zielregisters.
1	0	1	0	Direkte Angabe des Quelloperanden im Befehl, das Zielregister fungiert als Zeiger (Adresse) auf den Speicher.
1	0	1	1	Quelloperand ist eine Konstante, das Zielregister fungiert als Zeiger (Adresse) auf den Speicher.
1	1	0	x	Direkter Lesezugriff auf die Werte der Quell- und Zielregister. Das Ergebnis wird ins `ACCU`-Register geschrieben.
1	1	1	x	Direkter Lesezugriff auf den Wert des Quellregisters, das Zielregister fungiert als Zeiger (Adresse) auf den Speicher. Das Ergebnis wird ins `ACCU`-Register geschrieben.

Abkürzungsverzeichnis

ALFA	Automatische lineare Funktionsapproximation
ALU	Arithmetisch, logisches Rechenwerk (engl.: *arithmetic logic unit*)
ASIC	Anwendungsspezifische integrierte Schaltung (engl.: *application-specific integrated circuit*)
ASIP	Prozessor mit anwendungsspezifischem Befehlssatz (engl.: *application-specific instruction-set processor*)
AWGN	Gaußverteiltes, weißes Rauschen (engl.: *additive white gaussian noise*)
BER	Bitfehlerrate (engl.: *bit error rate*)
Bit	*Binary digit*
BS	Basisstation
CFO	Frequenzoffset (engl.: *carrier frequency offset*)
CMOS	*Complementary metal oxide semiconductor*
CORDIC	*Coordinate rotation digital computer*
CP	*Cyclic prefix*
CPU	Prozessorkern (engl.: *central processing unit*)
CRA	*Carry-Ripple*-Adder
CRAM	*Carry-Ripple-Array*-Multiplizierer
DFE	*Decision feedback equalization*
DRC	*Design-Rule-Check*
FDM	Frequenzmultiplex
FDMA	*Frequency division multiple access*
FFT	Schnelle Fouriertransformation (engl.: *fast fourier transform*)
FPGA	*Field programmable gate array*
FSM	*Finite state machine*
GE	*Gate equivalent*

GPIC	Mehrzweck-Schaltung (engl.: *general purpose integrated circuit*)
HA	Halbaddierer
HDL	*Hardware description language*
HLS	*High-Level*-Synthese
IC	Integrierte Schaltung (engl.: *integrated circuit*)
ICI	Trägerinterferenz (engl.: *inter carrier interference*)
IFFT	Inverse, schnelle Fouriertransformation (engl.: *inverse fast fourier transform*)
IO	*Input-Output*
IP	*Internet protocol*
ISI	Symbolinterferenz (engl.: *inter symbol interference*)
KNN	Künstliche neuronale Netze
KQ	(Methode der) kleinsten Quadrate
LISA	*Language for instruction set architectures*
LMS	*Least-Mean-Square*
LSI	*Large-Scale integration*
LTE	*Long term evolution*
LUT	Wertetabelle (engl.: *lookup-table*)
MAM	*Modified-Adams*-Methode
ML	*Maximum-Likelihood*
MLP	*Multilayer perceptron*
MIMO	*Multiple input multiple output*
MISO	*Multiple input single output*
MPSoC	Multiprozessor-System (engl.: *multi-processor system-on-chip*)
MSB	Höchstwertige(s) Bit(s) (engl.: *most significant bit(s)*)
MSI	*Medium-Scale Integration*
MU	Vielfachnutzung (engl.: *multi user*)
NCO	Numerisch gesteuerter Oszillator (engl.: *numerically controlled oscillator*)
OFDM	Orthogonales Frequenzmultiplexverfahren (engl.: *orthogonal frequency division multiplexing*)

PA	Polynomiale Approximation
PC	*Program counter*
QAM	Quadratur Amplituden Modulation
QF	Quantisierungsfaktor
RMSD	Wurzel der mittleren quadratischen Abweichung (engl.: *root mean square derivation*)
RRM	Reduzierter Radix-4 Multiplizierer
RTL	Register-Transfer Ebene (engl.: *register transfer level*)
RF	Radiofrequenz (engl.: *radio frequency*)
SA	Segmentbasierte Approximation
SDM	Raummultiplex (engl.: *space division multiplex*)
SFDR	*Spurious free dynamic range*
SIMO	*Single input multiple output*
SISO	*Single input single output*
SSI	*Small-Scale integration*
TCP	*Transmission control protocol*
THP	*Tomlinson-Harashima precoder*
ULSI	*Ultra-Large-Scale integration*
USART	Universeller, synchroner und asynchroner Transceiver (engl.: *universal synchronous and aynchronous serial receiver and transmitter*)
VA	Volladdierer
VHDL	*Very high speed integrated circuit hardware description language*
VLIW	*Very large instruction word*
VLSI	*Very-Large-Scale integration*
WLAN	*Wireless local area network*
ZF	*Zero-Forcing*

Begriffsklärung

Taktfrequenz
Der Begriff Taktfrequenz bezeichnet die durch den kritischen Pfad festgelegte maximale Betriebsfrequenz der Schaltung. Die Taktfrequenz verhält sich reziprok zur Taktdauer.

Taktdauer
Die Taktdauer (engl.: *delay*) ist die durch den kritischen Pfad gegebene zeitliche Verzögerung einer Schaltung. Sie verhält sich reziprok zur Taktfrequenz.

Datenpfad
Der Datenpfad kennzeichnet digitale Schaltungsteile, in denen ausschließlich die Verarbeitung von Nutzdaten, gegeben z.B. durch arithmetische Operanden, statt findet.

Entwurfskriterien
Der Begriff Entwurfskriterien (engl.: *constraints*) benennt die vorab spezifizierten Randbedingungen des Hardwareentwurfs. Entwurfskriterien lassen sich als entwicklungsorientierte Betrachtung von Randbedingungen verstehen, während die Performance anwendungsorientierten Charakter aufweist. Ein Überblick über gängige Kriterien ist in Kap. 2.4.1 gegeben.

Hardwareentwurf
Unter dem Bergiff Hardwareentwurf ist im Rahmen dieser Dissertation die Abarbeitungsreihenfolge von der Spezifikation zum Maskenlayout zu verstehen, nicht aber weitergehende Schritte wie Fertigung oder Testen.

Implementierung
Implementierung bezeichnet sämtliche Schritte zur hardwarebasierten Umsetzung eines Algorithmus oder einer Anwendung.

Komplexität
Die Komplexität gibt Aufschluss über die Größe eines Schaltungsentwurfs anhand von

Gatteräquivalenten, also der Anzahl sämtlicher verwendeten Gatter, normiert auf ein NAND-Gatter [55]. Dieses Verfahren erlaubt hinsichtlich der Bewertung und des Vergleichs unterschiedlicher Schaltungen eine gerechte Evaluation, da mögliche Technologieeinflüsse nicht, oder nur gering, zur Geltung kommen.

Latenz
Die Latenz beschreibt die durch synchrone Schaltelemente, z.B. Flipflops, auftretende taktbasierte Verzögerungen einer Schaltung.

Performance
Der Begriff Performance bewertet die erzielten Ergebnisse hinsichtlich relevanter Randbedingungen des Hardwareentwurfs für den Schaltungsentwurf. Die Performance lässt sich als anwendungsorientierte Betrachtung von Randbedingungen verstehen, während die Entwurfskriterien entwicklungsorientierten Charakter aufweisen. Im Rahmen dieser Arbeit beschränkt sich der Begriff Performance auf die technischen Randbedingungen Taktfrequenz, Latenz, Komplexität, Rechengenauigkeit und Energieverbrauch.

Programmierung
Programmierung bezeichnet sämtliche Schritte zur softwarebasierten Umsetzung eines Algorithmus oder einer Anwendung.

Steuerpfad
Der Steuerpfad kennzeichnet digitale Schaltungsteile, in denen ausschließlich die Ansteuerung und Konfiguration des Datenpfades statt findet.

Literaturverzeichnis

[1] S. Ahmadi: *An overview of next-generation mobile WiMAX technology.* In: *IEEE Communications Magazine*, Bd. 47, S. 84–98, Jun. 2009.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam und E. Cayirci: *Wireless Sensor Networks: A Survey.* In: *Computer Networks*, Bd. 38, S. 393–422, Jan. 2002.

[3] M. G. Arnold: *Approximating trigonometric functions with the laws of sines and cosines using the logarithmic number system.* In: *8th Euromicro Conference on Digital System Design*, S. 48–53, 2005.

[4] A. Ashrafi, R. Adhami und A. Milenkovic: *A Direct Digital Frequency Synthesizer Based on the Quasi-Linear Interpolation Method.* In: *IEEE Transactions on Circuits and Systems I: Regular Papers*, Bd. 57, S. 863–872, Apr. 2010.

[5] K. A. Atkinson: *An Introduction to Numerical Analysis.* Wiley and Sons, 1988.

[6] C. Behrens: *Kooperatives Energiemanagement in ressourcenbeschränkten Systemen.* Dissertation, Universität Bremen, 2009.

[7] O. Bischoff, N. Heidmann, J. Rust und S. Paul: *Design and Implementation of an Ultrasonic Localization System for Wireless Sensor Networks Using Angle-of-Arrival and Distance Measurement.* In: *The 26th European Conference on Solid-State Transducers (Eurosensors)*, S. 953–956, Sep. 2012.

[8] E. Bocchieri: *Automatic Speech Recognition on Mobile Devices and over Communication Networks.* Advances in Pattern Recognition. Springer-Verlag, 2008.

[9] R. P. Brent und H. T. Kung: *A Regular Layout for Parallel Adders.* In: *IEEE Transactions on Computers*, Bd. C-31, S. 260–264, Mär. 1982.

[10] I. N. Bronstein, K. A. Semendjaev, G. Musiol und H. Mühlig: *Taschenbuch der Mathematik.* Harri Deutsch, 6. Aufl., 2005.

[11] A. Burg: *VLSI Circuits for MIMO Communication Systems*. Dissertation, Swiss Federal Institute of Technology Zurich, 2006.

[12] J. R. Carson: *Reciprocal Theorems in Radio Communication*. In: *Proceedings of the Institute of Radio Engineers*, Bd. 17, S. 952–956, 1929.

[13] D. Chinnery und K. Keutzer: *Closing the Power Gap Between ASIC & Custom: Tools and Techniques for Low Power Design*. Springer-Verlag, 2007.

[14] V. K. Chippa, S. T. Chakradhar, K. Roy und A. Raghunathan: *Analysis and characterization of inherent application resilience for approximate computing*. In: *50th ACM / EDAC / IEEE Design Automation Conference (DAC)*, S. 1–9, 2013.

[15] T. D. Chiueh und P. Y. Tsai: *OFDM Baseband Receiver Design for Wireless Communications*. Wiley and Sons, 2007.

[16] T. D. Chiueh, P. Y. Tsai und I. W. Lai: *Baseband Receiver Design for Wireless MIMO-OFDM Communications*. Wiley and Sons, 2. Aufl., 2012.

[17] Y. J. Chong und S. Parameswaran: *Automatic Application Specific Floating-point Unit Generation*. In: *Design, Automation Test in Europe Conference Exhibition (DATE)*, S. 1–6, Apr. 2007.

[18] P. Christiansen: *Rechnergestütztes Entwickeln integrierter Schaltungen*. Vogel-Fachbuch, Schaltungstechnik. Vogel Verlag, 1989.

[19] Cisco Systems: *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012-2017*. Techn. Ber., Feb. 2013. http://www.cisco.com/.

[20] D. Clein und G. Simokurs: *CMOS IC Layout*. Elsevier Verlag, 2000.

[21] K. H. Cordes, A. Waag und N. Heuck: *Integrierte Schaltungen: Grundlagen - Prozesse - Design - Layout*. Person Studium, 2010.

[22] D. De Caro, N. Petra und A. G. M. Strollo: *A 380 MHz Direct Digital Synthesizer/Mixer With Hybrid CORDIC Architecture in 0.25 µm CMOS*. In: *IEEE Journal of Solid-State Circuits*, Bd. 42, S. 151–160, Jan. 2007.

[23] D. De Caro, N. Petra und A. G. M. Strollo: *Direct Digital Frequency Synthesizer Using Nonuniform Piecewise-Linear Approximation*. In: *IEEE Transactions on Circuits and Systems I: Regular Papers*, Bd. 58, S. 2409–2419, Okt. 2011.

[24] J. Detrey und F. de Dinechin: *A VHDL library of LNS operators*. In: *Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers, 2004.*, Bd. 2, S. 2227–2231, 2003.

[25] D. Djorković: *On the Hadamard product of matrices.* In: *Mathematische Zeitschrift*, Bd. 8, S. 395. Springer-Verlag, 1965.

[26] J. Duprat und J. M. Muller: *Hardwired polynomial evaluation.* In: *Journal of Parallel and Distributed Computing*, Bd. 5, S. 291–309, Jun. 1988.

[27] L. Fahrmeir, T. Kneib und S. Lang: *Regression: Modelle, Methoden und Anwendungen.* Springer-Verlag, 2. Aufl., 2009.

[28] W. Fan und C. S. Choy: *Power efficient and high speed frequency synchronizer design for MB-OFDM UWB.* In: *IEEE International Conference on Ultra-Wideband (ICUWB 2009)*, S. 669–673, Sep. 2009.

[29] D. Feng, C. Jiang, G. Lim, L. Cimini Jr, G. Feng und G. Li: *A survey of energy-efficient wireless communications.* In: *IEEE Communications Surveys Tutorials*, Bd. 15, S. 167–178, Feb. 2013.

[30] O. Forster: *Analysis 1: Differential- und Integralrechnung einer Veränderlichen.* Vieweg + Teubner Verlag, 9. Aufl., 2008.

[31] H. Foster: *Response checkers, monitors and assertions.* In: D. Pradhan und I. Harris (Hrsg.): *Practical Design Verification*, Kap. 4, S. 92–112. Cambridge University Press, Aug. 2009.

[32] D. D. Gajski und R. H. Kuhn: *Guest Editors' Introduction: New VLSI Tools.* Computer, 16:11–14, Dez. 1983.

[33] W. Gautschi: *Orthogonal Polynomials, Quadrature, and Approximation: Computational Methods and Software (in Matlab).* In: *Orthogonal Polynomials and Special Functions*, Bd. 1883 d. Reihe *Lecture Notes in Mathematics*, S. 1–77. Springer-Verlag, 2006.

[34] B. G. Goldberg: *Digital Frequency Synthesis Demystified.* Elsevier Inc., 1999.

[35] D. Goldberg: *What every computer scientist should know about floating-point arithmetic.* In: *ACM Computing Surveys*, Bd. 23, New York, NY, USA, Mär. 1991. ACM.

[36] G. H. Golub und C. F. Van Loan: *Matrix computations.* Johns Hopkins Univ. Press, 3. Aufl., 1996.

[37] V. Gupta, D. Mohapatra, A. Raghunathan und K. Roy: *Low-Power Digital Signal Processing Using Approximate Adders.* In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Bd. 32, S. 124–137, Jan. 2013.

[38] R. Habendorf: *Vorentzerrung für die räumlich überlagerte Kommunikation mit verteilten Empfängern*. Dissertation, Technische Universität Dresden, 2008.

[39] J. Han und M. Orshansky: *Approximate computing: An emerging paradigm for energy-efficient design*. In: *18th IEEE European Test Symposium (ETS)*, S. 1–6, 2013.

[40] C. Haubelt und J. Teich: *Digitale Hardware/Software-Systeme: Spezifikation und Verifikation*. Springer-Verlag, 2010.

[41] S. Haykin: *Neural Networks and Learning Machines*. Prentice Hall, 3. Aufl., 2009.

[42] M. Hübner und J. Becker: *Multiprocessor System-on-Chip: Hardware Design and Tool Integration*. Springer-Verlag, 2011.

[43] N. Heidmann, T. Wiegand und S. Paul: *Architecture and FPGA-implementation of a high throughput K^+-Best detector*. In: *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, S. 1–6, 2011.

[44] J. L. Hennessy und D. A. Patterson: *Computer Architecture: A quantitative approach*. Morgan Kaufman, 2. Aufl., 1996.

[45] J. D. Hoffman: *Numerical Methods for Engineers and Scientists*. McGraw-Hill, 1992.

[46] X. Hu, R. G. Harber und S. C. Bass: *Expanding the range of convergence of the CORDIC algorithm*. In: *IEEE Transactions on Computers*, Bd. 40, S. 13–21, Jan. 1991.

[47] Z. Y. Huang und P. Y. Tsai: *Efficient Implementation of QR Decomposition for Gigabit MIMO-OFDM Systems*. In: *IEEE Transactions on Circuits and Systems I: Regular Papers*, Bd. 58, S. 2531–2542, Okt. 2011.

[48] J. Hubbard, D. Schleicher und S. Sutherland: *How to Find All Roots of Complex Polynomials by Newton's Method*. In: *Inventiones Mathematicae*, Bd. 146, S. 1–33, 2000.

[49] R. J. Hyndman und A. B. Koehler: *Another look at measures of forecast accuracy*. In: *International Journal of Forecasting*, Bd. 22, S. 679–688, 2006. http://www.sciencedirect.com/.

[50] IEEE: *IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis*, 2004. http://ieeexplore.ieee.org/servlet/opac?punumber=6748.

[51] IEEE: *IEEE Standard for Verilog Hardware Description Language*, 2006. http://ieeexplore.ieee.org/servlet/opac?punumber=10779.

[52] IEEE: *IEEE Standard for Floating-Point Arithmetic*, 2008. http://ieeexplore.ieee.org/servlet/opac?punumber=4610933.

[53] G. Irvine, L. Wang, P. Dickman und D. R. S. Cumming: *Variable-rate data sampling for low-power microsystems using modified Adams methods.* In: *IEEE Transactions on Signal Processing*, Bd. 51, S. 3182–3190, Dez. 2003.

[54] R. Kacimi, R. Dhaou und A. L. Beylot: *Using Energy-Efficient Wireless Sensor Network for Cold Chain Monitoring.* In: *6th IEEE Consumer Communications and Networking Conference (CCNC)*, S. 1–5, 2009.

[55] H. Kaeslin: *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication.* Cambridge University Press, 2008.

[56] K. D. Kammeyer: *Nachrichtenübertragung.* B.G. Teubner Verlag, 3. Aufl., 2004.

[57] H. Karl und A. Willig: *Protocols and Architectures for Wireless Sensor Networks.* Wiley and Sons, 2005.

[58] M. Keating, D. Flynn, R. Aitken und A. G. K. Shi: *Low Power Methodology: For System-on-Chip Design.* Series on integrated circuits and systems. Springer-Verlag, 2008.

[59] W. Kester: *Understand SINAD, ENOB, SNR, THD, THD + N, and SFDR so You Don't Get Lost in the Noise Floor.* Techn. Ber., Analog Devices, 2011. http://www.analog.com.

[60] M. Khanafer, M. Guennoun und H. Mouftah: *WSN Architectures for Intelligent Transportation Systems.* In: *New Technologies, Mobility and Security (NTMS), 2009 3rd International Conference on*, S. 1–8, Dez. 2009.

[61] J. S. Kim und M. Sunwoo: *Three low power ASIP processor designs for communications, video, and audio applications.* In: *International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, S. 241–244, Sep. 2007.

[62] M. Knorrenschild: *Numerische Mathematik: Eine beispielorientierte Einführung.* Hanser, 5. Aufl., 2013.

[63] D. E. Knuth: *The Art of Computer Programming: Seminumerical Algorithms*, Bd. 2. Addison Wesley, 3. Aufl., 1998.

[64] P. M. Kogge und H. S. Stone: *A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations*. In: *IEEE Transactions on Computers*, Bd. C-22, S. 786–793, Aug. 1973.

[65] N. Kollerstrom: *Thomas Simpson and 'Newton's Method of Approximation': An Enduring Myth*. In: *The British Journal for the History of Science*, Bd. 25, S. 347–354. Spe. 1992.

[66] S. Y. Kung: *Digital Neural Networks*. Prentice Hall, Mär. 1993.

[67] W. K. Lam: *Hardware design verification: simulation and formal method-based approaches*. Prentice Hall, 2005.

[68] W. Lang, R. Jedermann, D. Mrugala, A. Jabbari, B. Krieg-Brückner und K. Schill: *The "Intelligent Container" ;A Cognitive Sensor Network for Transport Management*. In: *IEEE Sensors Journal*, Bd. 11, S. 688–698, Mär. 2011.

[69] M. B. Lin: *Digital system design and practices using Verilog HDL and FPGAs*. Wiley and Sons, 2008.

[70] D. Lister: *An operator's view on green radio*. Präsentation auf dem *First International Workshop on Green Communications (GreenCom)*, 2009 http://www.green-communications.net/icc09/.

[71] H. Liu und G. Li: *OFDM-based Broadband Wireless Networks: Design and Optimization*. Wiley and Sons, 2005.

[72] P. Luethi, C. Studer, S. Duetsch, E. Zgraggen, H. Kaeslin, N. Felber und W. Fichtner: *Gram-Schmidt-based QR decomposition for MIMO detection: VLSI implementation and comparison*. In: *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, S. 830–833, Dez. 2008.

[73] K. Mallinson: *2020 Vision for LTE*, 2013. http://www.fiercewireless.com/.

[74] W. Mangione-Smith, B. Hutchings, D. Andrews, A. DeHon, C. Ebeling, R. Hartenstein, O. Mencer, J. Morris, K. Palem, V. Prasanna und H. Spaanenburg: *Seeking solutions in configurable computing*. In: *Computer*, Bd. 30, Aug. 1997.

[75] M. M. Mano: *Digital Design*. Prentice Hall, 3. Aufl., 2002.

[76] P. Marwedel: *Eingebettete Systeme*. Springer-Verlag, Berlin, Heidelberg, 2008.

[77] P. Meher, J. Valls, T. B. Juang, K. Sridharan und K. Maharatna: *50 Years of CORDIC: Algorithms, Architectures, and Applications*. In: *IEEE Transactions on Circuits and Systems I: Regular Papers*, Bd. 56, S. 1893–1907, Sep. 2009.

[78] D. Miorandi, S. Sicari, F. D. Pellegrini und I. Chlamtac: *Internet of things: Vision, applications and research challenges.* In: *Ad Hoc Networks*, Bd. 10, S. 1497–1516. Sep. 2012.

[79] J. N. Mitchell: *Computer Multiplication and Division Using Binary Logarithms.* In: *IRE Transactions on Electronic Computers*, Bd. EC-11, S. 512–517, Aug. 1962.

[80] M. Morales-Sandoval, C. Feregrino-Uribe, R. Cumplido und I. Algredo-Badillo: *A reconfigurable GF(2^M) elliptic curve cryptographic coprocessor.* In: *VII Southern Conference on Programmable Logic (SPL)*, S. 209–214, 2011.

[81] MOTEIV: *TMotesky Datasheet*, Nov. 2006.

[82] J. M. Muller: *Elementary Functions: algorithms and implementation.* Birkhäuser Bosten, 2. Aufl., 2005.

[83] J. Nurmi: *Processor Design: System-on-Chip computing for ASICs and FPGAs.* Springer-Verlag, 2007.

[84] B. Parhami: *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford Press University, Inc., 2. Aufl., 2009.

[85] T. Parr: *Stringtemplate 3.0 Documentation*, 2008. http://www.stringtemplate.org.

[86] D. Pathel, M. Shabany und P. Genn Gulak: *A low-complexity high-speed QR decomposition implementation for MIMO receivers.* In: *IEEE International Symposium on Circuits and Systems (ISCAS)*, S. 33–36, 2009.

[87] S. Pees, A. Hoffmann, V. Zivojnovic und H. Meyr: *LISA-machine description language for cycle-accurate models of programmable DSP architectures.* In: *36th Design Automation Conference (DAC)*, S. 933–938, 1999.

[88] M. Petermann: *Robuste Vorentzerrung in TDD-Systemen mit nicht perfekter Kanalreziprozität.* Dissertation, Universität Bremen, 2012.

[89] D. Pham, S. Asano, M. Bolliger, M. Day, H. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki und K. Yazawa: *The design and implementation of a first-generation CELL processor.* In: *IEEE International Solid-State Circuits Conference (ISSCC)*, S. 184–592, 2005.

[90] P. Pirsch: *Architekturen der digitalen Signalverarbeitung.* Teubner, 5. Aufl., 1996.

[91] M. Platzner, J. Teich und N. Wehn: *Dynamically Reconfigurable Systems: Architectures, Design Methods and Applications.* Springer-Verlag, 2010.

[92] J. G. Proakis und M. Salehi: *Digital Communications.* McGraw-Hill, 2008.

[93] J. Rabaey: *Low Power Design Essentials.* Springer-Verlag, 2009.

[94] V. Raghunathan, C. Schurgers, S. Park und M. Srivastava: *Energy-aware wireless microsensor networks.* In: *IEEE Signal Processing Magazine*, Bd. 19, S. 40–50, Mär. 2002.

[95] D. Raychaudhuri und N. B. Mandayam: *Frontiers of Wireless and Mobile Communications.* In: *Proceedings of the IEEE*, Bd. 100, S. 824–840, Apr. 2012.

[96] J. Richling, M. Werner, M. Jaeger, G. Mühl und H. U. Heiß: *Autonomie in verteilten IT-Architekturen.* Oldenbourg, 2011.

[97] F. Rosenblatt: *The Perceptron: A probabilistic model for information storage and organization in the brain.* Psychological Review, 65:386–408, 1958.

[98] M. Rosenlicht: *Liouville's Theorem on Functions with Elementary Integrals.* In: *Pacific Journal of Mathematics*, Bd. 24, S. 153–161, 1968.

[99] C. Roth, A. Cevrero, C. Studer, Y. Leblebici und A. Burg: *Area, throughput, and energy-efficiency trade-offs in the VLSI implementation of LDPC decoders.* In: *IEEE International Symposium on Circuits and Systems (ISCAS)*, S. 1772–1775, 2011.

[100] J. Rust, F. Ludwig und S. Paul: *Low Complexity QR-Decomposition Architecture Using the Logarithmic Number System.* In: *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, S. 97–102, 2013.

[101] J. Rust, C. Osewold und S. Paul: *Implementation of a Low Power Low Complexity ASIP for various Sphere Decoding Algorithms.* In: *11th European Wireless Conference (European Wireless)*, S. 1–6, 2011.

[102] J. Rust und S. Paul: *Design and Implementation of a Neurocomputing ASIP for environmental monitoring in WSN.* In: *19th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, S. 129 – 132, 2012.

[103] J. Rust und S. Paul: *A Direct Digital Frequency Synthesizer Based on Automatic Nonuniform Piecewise Function Generation.* In: *Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, S. 230–234, 2012.

[104] J. Rust, X. Wang, M. Gröning, R. Laur und S. Paul: *Implementation of an Artificial Neural Network Hardware Accelerator for Environmental Monitoring in Wireless Sensor Networks.* In: *2011 Seventh International Conference on Networked Sensing Systems (INSS)*, S. 1–6, 2011.

[105] J. Rust, X. Wang, R. Laur und S. Paul: *A High Performance Neurocomputing Algorithm for Prediction Tasks in Wireless Sensor Networks.* In: *4th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, S. 1–5, 2011.

[106] J. Rust, X. Wang, R. Shen, R. Laur und S. Paul: *Equidistant Piecewise function Approximation for neurocomputing based environmental monitoring in wireless sensor networks.* In: *IEEE Workshop on Environmental Energy and Structural Monitoring Systems (EESMS)*, S. 1–5, 2011.

[107] J. Rust, T. Wiegand und S. Paul: *Design and Implementation of a Low Complexity NCO based CFO Compensation Unit.* In: *Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, S. 116–120, 2012.

[108] N. Sapankevych und R. Sankar: *Time Series Prediction Using Support Vector Machines: A Survey.* In: *IEEE Computational Intelligence Magazine*, Bd. 4, S. 24–38, Mai 2009.

[109] O. Schliebusch, H. Meyr und R. Leupers: *Optimized ASIP Synthesis from Architecture Description Language Models.* Springer-Verlag, 2007.

[110] P. Sebah und X. Gourdon: *Newton's method and high-order iterations.* Techn. Ber., Hungarian Academy of Sciences, 2001. http://www.sztaki.hu/.

[111] L. F. Shampine und M. K. Gordon: *Computer solution of ordinary differential equations.* W.H.Freeman & Co Ltd, 1975.

[112] K. Sohraby, D. Minoli und T. Znati: *Wireless Sensor Networks: Technology, Protocols, and Applications.* Wiley and Sons, 2007.

[113] C. Studer, P. Blosch, P. Friedli und A. Burg: *Matrix Decomposition Architecture for MIMO Systems: Design and Implementation Trade-offs.* In: *Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers, 2007. ACSSC 2007.*, S. 1986–1990, 2007.

[114] Y. Sun und M. S. Kim: *A High-Performance 8-Tap FIR Filter Using Logarithmic Number System.* In: *IEEE International Conference on Communications (ICC), 2011*, S. 1–5, 2011.

[115] L. K. Tan und H. Samueli: *A 200-MHz quadrature digital synthesizer/mixer in 0.8- mu;m CMOS*. In: *Proceedings of the IEEE 1994 Custom Integrated Circuits Conference, 1994.*, Bd. 30, S. 193–200, Aug. 1995.

[116] A. S. Tanenbaum: *Computerarchitektur: Strukturen, Konzepte, Grundlagen*. Prentice Hall, 5. Aufl., 2005.

[117] A. S. Tanenbaum und M. v. Steen: *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2. Aufl., 2006.

[118] Texas Instruments: *MSP430x1xx Family User's Guide (Rev. F)*, 2006. http://www.ti.com.

[119] UMC: *UMC Free Library*, 2013. http://freelibrary.faraday-tech.com/.

[120] J. E. Volder: *The CORDIC Trigonometric Computing Technique*. In: *IRE Transactions on Electronic Computers*, Bd. EC-8, Sep. 1959.

[121] R. Walker und D. Thomas: *A Model of Design Representation and Synthesis*. In: *22nd Conference on Design Automation (DAC)*, S. 453–459, 1985.

[122] J. S. Walther: *A unified algorithm for elementary functions*. In: *Proceedings of the May 18-20, 1971, spring joint computer conference*, AFIPS '71 (Spring), S. 379–385, Mai 1971.

[123] L. T. Wang, Y. W. Chang und K. T. Cheng: *Electronic design automation: synthesis, verification, and test*. The Morgan Kaufmann series in systems on silicon. Elsevier, Morgan Kaufmann, 2009.

[124] X. Wang, A. Jabbari, R. Jedermann, R. Laur und W. Lang: *Adaptive data sensing rate in ad-hoc sensor networks for autonomous transport application*. In: *13th Conference on Information Fusion (FUSION)*, S. 1–8, 2010.

[125] D. Wübben: *Effiziente Detektionsverfahren für Multilayer-MIMO-Systeme*. Dissertation, Universität Bremen, 2005.

[126] N. H. E. Weste und D. M. Harris: *Integrated Circuit Design*. Prentice Hall, 4. Aufl., 2010.

[127] D. Widmann, H. Mader und H. Friedrich: *Technologie hochintegrierter Schaltungen*. Springer-Verlag, 1988.

[128] T. Wiegand: *Hardwareoptimierte Entwicklung nachichtentechnischer Algorithmen zum Prototypenentwurf von Mobilfunkempfängern*. Dissertation, Universität Bremen, 2012.

[129] T. Wiegand und S. Paul: *Reduced complexity computation unit for a sphere decoding algorithm.* In: *18th European Wireless Conference (European Wireless)*, S. 1–6, 2012.

[130] C. Windpassinger: *Detection and Precoding for Multiple Input Multiple Output Channels.* Dissertation, Universität Erlangen-Nürnberg, 2004.

[131] J. Zyren und W. McCoy: *Overview of the 3GPP long term evolution.* Techn. Ber., Freescale Semiconductor, Inc, 2007. http://www.freescale.com/.