Fachhochschule Westküste
Hochschule für Wirtschaft & Technik

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Fabian Zahn

Entwicklung eines Verfahrens zur Spektralschätzung für die Diagnose von ABS-Sensorsignalen

Fachhochschule Westküste
Fachbereich Technik

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

Hamburg University of Applied Sciences
Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Fabian Zahn

Entwicklung eines Verfahrens zur Spektralschätzung für die Diagnose von ABS-Sensorsignalen

Masterarbeit eingereicht im Rahmen der Masterprüfung
im gemeinsamen Masterstudiengang Mikroelektronische Systeme
am Fachbereich Technik
der Fachhochschule Westküste
und
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Karl-Ragmar Riemschneider
Zweitgutachter : Prof. Dr. Stephan Hußmann

Abgegeben am 5. April 2013

Fabian Zahn

Thema der Masterarbeit

Entwicklung eines Verfahrens zur Spektralschätzung für die Diagnose von ABS-Sensorsignalen

Stichworte

Digitale Signalverarbeitung, Rademacher-Funktionen, Spektralschätzung, DFT, ESZ-ABS, ABS, AMR, VHDL

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Thematik der ABS-Sensoren deren Funktion um die Selbstdiagnose erweitert werden soll. Für diesen Zweck wird ein Verfahren entworfen, das die spektralen Anteile in einem ABS-Sensorsignal schätzt und somit die bisher eingesetzte diskrete Fourier-Transformation ersetzt. Das entworfene Verfahren wird zusätzlich ausführlich getestet, exemplarisch in VHDL implementiert und im Hinblick auf eine CMOS-Chip Implementierung analysiert und bewertet.

Fabian Zahn

Title of the paper

Development of a spectral estimation method for diagnosis of ABS sensor signals

Keywords

Digital Signal Processing, Rademacher-Functions, Spectrum Estimation, DFT, ESZ-ABS, ABS, AMR, VHDL

Abstract

This work deals with the issue of the ABS sensors with the function for self-diagnosis. For this purpose a method is designed to estimate the spectral components in an ABS-sensor signal in order to replace the previously used discrete Fourier-transform. The developed spectral estimation method is also tested extensively, exemplified implemented in VHDL and analyzed in terms of a envisioned CMOS chip implementation.

”A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools”

– Douglas Adams

Danksagung

An erster Stelle danke ich meiner Familie für ihre Unterstützung während des gesamten Studiums. Nicht zu vergessen danke ich all den Kommilitonen, die im Laufe der letzten Jahre zu sehr guten Freunden geworden sind und deren Hilfsbereitschaft und Unterstützung einen nennenswerten Beitrag zum erfolgreichen Abschluss dieses Studiums geleistet haben.

Besonderer Dank gilt Herrn Dipl.-Ing. (FH) Martin Krey für seinen fachlichen Rat und seine tatkräftige Unterstützung. Außerdem bedanke ich mich bei dem gesamten Forschungsteam der ESZ-ABS und BATSEN Projekte für das angenehme und kollegiale Arbeitsklima.

Meinen Betreuern Prof. Dr. Karl-Ragmar Riemschneider und Prof. Dr. Stephan Hußmann gilt besonderer Dank für ihre Zeit, Geduld und konstruktiven Anregungen.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

ABS-Sensoren und die Erkennung des Zustandes

Aus heutigen PKW und LKW sind Antiblockiersysteme kaum noch wegzudenken. Diese technischen Systeme tragen in höchstem Maße zur besseren Fahrsicherheit durch das Verhindern von Radblockierungen im Fahrbetrieb bei und ermöglichen z.B. ein besseres Lenk- und Spurverhalten des Fahrzeugs.

Im Rahmen des vom Bundesministerium für Bildung und Forschung (BMBF) geförderten Projektes *Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren* (kurz ESZ-ABS) soll die Zuverlässigkeit von ABS-Sensorsystemen durch die Zustandserkennung und die Funktion zur Selbstdiagnose erhöht werden. Moderne ABS-Sensoren sind jedoch bisher nicht mit den angestrebten Fähigkeiten zur Selbstdiagnose ausgestattet. Dies macht es unmöglich im laufenden Betrieb das korrekte Verhalten des Sensors zu verifizieren.

Die am dafür eingesetzten Sensoren basieren entweder auf dem Hall-Effekt oder dem *anisotropen magnetoresistiven* Effekt, dem AMR-Effekt. Auf der Radnabe wird für die Erfassung des Radzustandes ein Encoderrad montiert, das ein zeitveränderliches Magnetfeld moduliert. Dieses Feld kann von dem Sensor aufgenommen und ausgewertet werden wie die Abbildung 1.1 zeigt.

Der im Projekt ESZ-ABS verwendete Sensortyp nutzt den anisotropen magnetoresistiven Effekt. Dieser beruht auf der Tatsache, dass sich die resistiven Eigenschaften gewisser ferromagnetischer Materialien in Abhängigkeit zu der Raumrichtung des angelegten Magnetfeldes ändern.

Der elektrische Widerstand wird am größten, wenn das äußere Magnetfeld *in* oder *gegen* die Stromrichtung gerichtet ist. Am kleinsten wird dieser Widerstand hingegen, wenn das Magnetfeld senkrecht zur Stromrichtung steht (siehe [6]).

Für Sensoranwendungen kann dieses Prinzip sehr einfach durch die sogenannte Wheatstonesche Messbrückenschaltung genutzt werden, indem man vier solcher magneto-resistiven Elemente speziell geometrisch anordnet, und diese entsprechend dem Magnetfeld aussetzt.

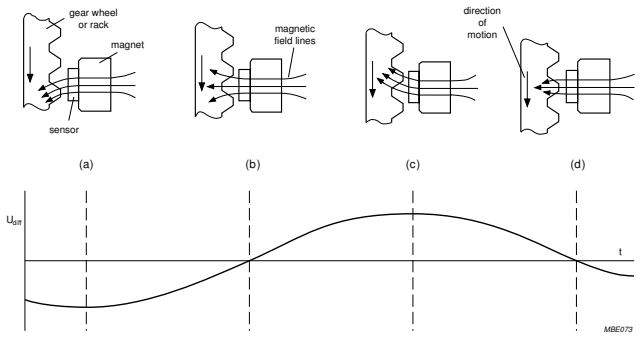


Abbildung 1.1.: In Abhängigkeit der Bewegung (bzw. der Position) des Encoderrades wird ein Magnetfeld moduliert, welches von dem Sensor aufgenommen werden kann. Es entsteht so im Idealfall eine sinusförmige Brückenspannung U_{diff} (Abbildung entnommen aus [21]).

Nachteilig ist jedoch, dass diese Sensoren ein sehr ausgeprägtes nichtlineares Übertragungsverhalten aufweisen. Es kann jedoch genutzt werden, um auf eine Aussage über den Arbeitspunkt des Sensors zu schließen. Diese nichtlineare Kennlinie kann in gewissen Arbeitspunkten für Verzerrungen in Form von weiteren harmonischen Anteilen im Spektrum des Nutzsignals verantwortlich sein. Die enstehenden Verzerrungen können so stark werden, dass das darunterliegende Signal kaum noch detektierbar ist.

Es ist das Ziel des Projektes eben diese Verzerrungen zu detektieren, um einen ABS-Sensor mit integrierter Selbstdiagnosefunktionalität zu entwerfen. Dieser soll in Zukunft in das Mikrosystem des Sensors integriert werden (siehe Abbildungen 1.2 und 1.3). Für diesen Zweck muss eine sehr präzise und zudem effizient implementierbare Lösung entstehen, die auf modernen CMOS-Prozessen eingesetzt werden kann, um die Diagnosefunktionalität direkt zu integrieren.

Diese Masterarbeit soll zur Aufwandsminimierung der Schätzung spektraler Anteile des Sensorsignals beitragen, da die Schätzung von harmonischen Anteilen im Nutzsignal des ABS-Sensors die Basis für die Selbstdiagnosefunktionalität darstellt. Zu diesem Zweck wird ein Verfahren zur Spektralschätzung auf der Basis von Rechteckfunktionen eingeführt. Dieses verfolgt die ehrgeizigen Ziele einerseits die benötigte Chipfläche zu reduzieren und andererseits die Systemtaktfrequenz herabzusetzen.

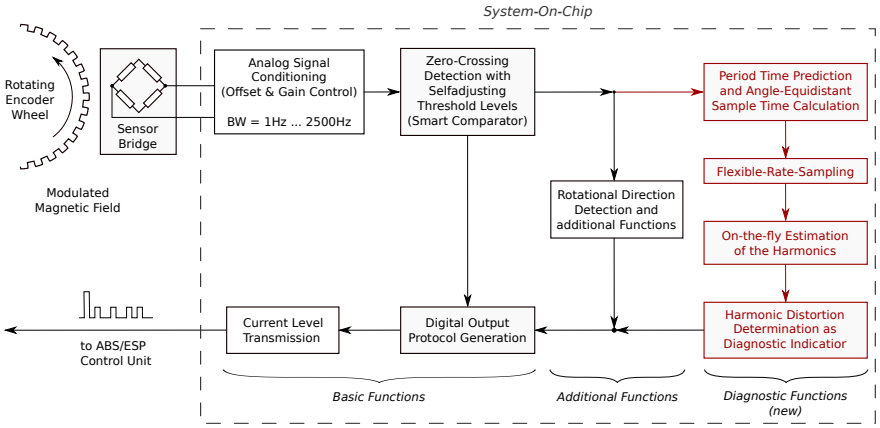


Abbildung 1.2.: Exemplarische Darstellung des angestrebten ABS-Sensors mit Diagnosefunktion. Die rot gekennzeichneten Bereiche stellen die Erweiterungen aus dem Projekt ESZ-ABS dar (basierend auf [26]).

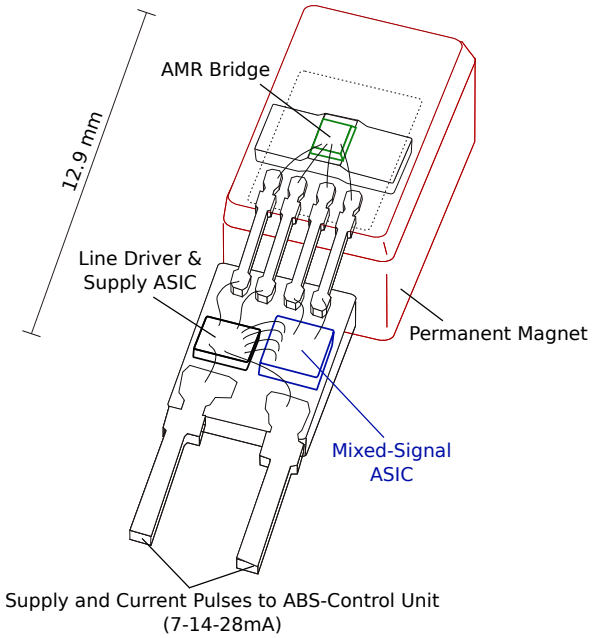


Abbildung 1.3.: Diese Darstellung zeigt das Mikrosystem eines ABS-Sensors. Mit der Integration der Funktion zur Selbstdiagnose in den Mixed-Signal ASIC soll im Forschungsprojekt ESZ-ABS ein Schritt in die Richtung zukünftiger Sensorgenerationen getan werden. Abbildung entnommen aus [25] und modifiziert.

1.1. Aufbau dieser Masterarbeit

In den folgenden Kapiteln wird die Entwicklung eines Verfahrens zur Spektralschätzung von ABS-Sensorsignaldaten beschrieben. Hierzu werden die bisherigen Verfahren und Ideen des Forschungsprojektes kurz zusammengefasst und anschließend als Basis für die Entwicklung des Verfahrens genutzt.

Darrauffolgend wird der entwickelte Algorithmus umfassenden Tests unterzogen, die die Anwendbarkeit des Verfahrens belegen sollen. Zu diesem Zweck werden unterschiedliche Signaltypen aus Simulations- und Messdaten genutzt, die eine Bewertung der Tauglichkeit des Verfahrens ermöglichen.

Im abschließenden Teil der Arbeit wird eine exemplarische Hardwareimplementierung aufgezeigt. Diese wird mit der bisherigen Implementierung in Bezug auf die benötigten Ressourcen kritisch verglichen und bewertet. Ferner werden die Limitierungen des implementierten Verfahrens herausgearbeitet und analysiert.

Kapitel 1 - Einleitung
Das erste Kapitel gibt eine kurze Einführung in die Thematik der auf dem AMR-Effekt basierenden ABS-Sensoren.

Kapitel 2 - Grundlagen
In diesem Kapitel werden die für das Verständnis notwendigen Grundlagen kurz erklärt und zusammengefasst.

Kapitel 3 - Analyse
Die Analyse befasst sich mit der Untersuchung der bisherigen Arbeiten im Projekt ESZ-ABS und dem Ausgangspunkt dieser Arbeit. Zusätzlich wird die erste Testchip Implementierung in Bezug auf die Fläche analysiert. Zudem werden die bisherigen Indikatoren für das Verzerrungsmaß betrachtet, und eine Abgrenzung zu anderen Verfahren der Spektralanalyse bzw. Spektralschätzung gegeben. Abschließend wird das Ziel dieser Arbeit herausgearbeitet.

Kapitel 4 - Konzept und Entwurf des Verfahrens
Dieses Kapitel setzt sich mit dem Entwurf des Verfahrens auf mathematischer Ebene auseinander und beschreibt die Entstehung des Algorithmus bzw. die Ideen zur Fehlerkompensation.

Kapitel 5 - Verifikation der Anwendbarkeit des Algorithmus
Zur Verifikation der Anwendbarkeit des entworfenen Algorithmus werden in diesem Kapitel diverse Simulationen in MATLAB durchgeführt, die den Algorithmus anhand von Monte-Carlo Simulationen und Simulationen mit realen Messdaten evaluieren.

Kapitel 6 - Exemplarische Implementierung
Anhand einer exemplarischen Realisierung wird das in Kapitel 4 vorgestellte Verfahren implementiert und evaluiert.

Kapitel 7 - Fazit und Ausblick
Im letzten Kapitel werden eine Zusammenfassung der Ergebnisse sowie ein Ausblick mit Ideen und Motivationsanregungen für weitere Projekte gegeben.

1.2. Anmerkung

In dieser Arbeit werden häufig englische Termini für die entsprechenden Fachbegriffe übernommen, da deren deutsche Übersetzungen einerseits nicht im Sprachgebrauch der Domäne akzeptiert werden und andererseits einige technische Begriffe nicht übersetzbar sind ohne ihre Bedeutung zu verlieren.

Außerdem sei an dieser Stelle angemerkt, dass sämtliche eigene Graphiken und Diagramme, die in dieser Arbeit verwendet werden, im Hinblick auf die Weiterverwendung bzw. für Publikationszwecke in englischer Sprache verfasst sind.

Weiterhin sei drauf hingewiesen, dass das in dieser Arbeit verwendete generische Maskulinum weibliche Personen in allen Zusammenhängen ausnahmslos mit einschließt.

2. Grundlagen

Dieses Kapitel gibt eine kurze Einführung in die im Forschungsprojekt ESZ-ABS genutzten Begrifflichkeiten und Algorithmen, die für das Verständnis der Entwicklung des Spektralschätzungsverfahrens essentiell sind. Die wichtigsten Begriffe sind kurz zusammengefasst und die Algorithmen bzw. deren Berechnung möglichst kompakt erklärt.

Hierzu ist es nützlich sich weiterer Sekundärliteratur aus dem Projekt ESZ-ABS zu bedienen, um sich vertiefend in die Thematik der auf dem AMR-Effekt basierten ABS-Sensorsysteme einzuarbeiten.

Wie bereits in der Einleitung erwähnt, ist es für die angestrebte Selbstdiagnosefunktion notwendig, die Signalqualität der Messbrückendifferenzspannung des ABS-Sensors zu analysieren. In den vorangegangen Arbeiten wurde gezeigt, dass dies sehr gut mittels sogenannter Verzerrungsmaße möglich ist.
Diese Verzerrungsmaße werden aus dem Spektrum des Signals berechnet, indem die harmonischen Anteile des Signals analysiert werden, wie es z.B. im Audiobereich für die Bestimmung des Klirrfaktors üblich ist.

2.1. Bestimmung des Verzerrungsmaßes

In den vorigen Arbeiten im Projekt ESZ-ABS wurde gezeigt, dass der Zustand sehr gut über die harmonischen Verzerrungen des Sensorsignals bestimmt werden kann. Auf die Korrektheit der Sensorfunktion (bzw. die des Arbeitspunktes) wird anhand der harmonischen Verzerrungen des Messbrückensignals geschlossen, indem aus dem Spektrum des Nutzsignals Indikatoren für das Verzerrungsmaß gebildet werden.

Dazu werden momentan verschiedene Indikatoren genutzt. Im Nachfolgenden soll erklärt werden, welche Indikatoren genutzt werden und wie sie berechnet werden können.

Im Allgemeinen wird der Ansatz gewählt, die gesamte harmonische Verzerrung (Total Harmonic Distortion: THD) des Signals zu bestimmen, um Aufschluss über den Sensorzustand zu erhalten. Dies kann man analog zur Klirrfaktormessung im Audiobereich betrachten. Die Qualität eines Signals wird anhand der Signalleistung des Nutzsignals und der im Signal vorhandenen Oberwellen bestimmt.

Im Verlauf des Projektes zeigte sich, dass zwei unterschiedliche Methoden zur Bestimmung der Verzerrungen besonders gut für den Anwendungsfall bei ABS-Sensoren geeignet sind.

2.1.1. Harmonic Distortion Infinite (HDI)

Als Harmonic Distortion Infinite (HDI) wird im Projekt ESZ-ABS die Berechnung eines Verzerrungsmaßes über das Verhältnis der Oberwellen bezogen auf die Gesamtwechselleistung des Signals bezeichnet. Dieses Verfahren wurde von Koch in [12] ausführlich besprochen und definiert. Der HDI wird üblicherweise prozentual dargestellt und kann durch folgende Gleichung beschrieben werden:

$$\mathrm{HDI}_{\%} = 100 \cdot \sqrt{\frac{\sum_{i=2}^{\infty} P_i}{\sum_{i=1}^{\infty} P_i}}\ \% \tag{2.1}$$

P_i bezeichnet die Leistungsspektrallinien des Eingangssignals und P_1 stellt die Leistung der ersten Harmonischen (also die der Grundwelle selbst) dar.

Für die Berechnung dieser Leistungsspektrallinien kann die DFT genutzt werden, indem man aus dem Amplitudenspektrum die Effektivwerte der Spannungen bestimmt und diese über einem Normierungswiderstand von $R = 1$ auswertet.

$$P_i = \frac{{U_{\mathrm{eff,i}}}^2}{R} \quad \text{mit} \quad U_{\mathrm{eff,i}} = \frac{\hat{u}_i}{\sqrt{2}} \quad \text{und} \quad R = 1 \quad \text{folgt} \tag{2.2}$$

$$P_i = \frac{\hat{u}_i^2}{2} \tag{2.3}$$

Den Scheitelwert $\hat{u}$ erhalten wir aus dem einseitigen Betragsspektrum, welches über die DFT gebildet wird. Dafür müssen wir den erhaltenen Wert lediglich mit einem Faktor von zwei multiplizieren.

$$\hat{u} = 2 \cdot h_i \quad \text{mit} \quad h_i = \frac{1}{N} \cdot |\mathrm{DFT}(u)_i| \tag{2.4}$$

$$P_i = 2 \cdot h_i^2 \tag{2.5}$$

Die Bestimmung der Wechselleistung des Signals kann unter Verwendung einer N-Punkte DFT erfolgen. Alternativ dazu ist es jedoch auch möglich, unter Verwendung von Parsevals Theorem (bzw. des Rayleigh Energie Theorems) (siehe [30]) die gesamte Wechselleistung

des Signals aus dem Zeitbereich direkt zu berechnen. Dieses Theorem besagt nämlich, dass immer gilt:

$$\sum_{n=0}^{N-1} |x_n|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X_k|^2 \tag{2.6}$$

wobei X_k die diskrete Fourier-Transformierte von x_n ist (jeweils der Länge N). Folglich kann man sowohl aus dem Zeitsignal als auch aus dem diskreten Spektrum des Signals das HDI Verzerrungsmaß berechnen. In der Praxis bietet es sich an das Zeitsignal zu verwenden, da für diesen Zweck nur eine Spektrallinie, nämlich die der Grundwelle, berechnet werden muss.

Die Bestimmung des HDI Verzerrungsmaßes in einem technischen System kann nun wie folgt durchgeführt werden:

$$\mathrm{HDI}_{\%} = 100 \cdot \sqrt{\frac{P_{total} - P_1}{P_{total}}}\ \% \tag{2.7}$$

$$P_{total} = P_{ms} - P_{dc} \tag{2.8}$$
$$:= \text{Gesamte Wechselleistung des Signals}$$

$$P_{ms} = \frac{1}{N} \cdot \left(\sum_{n=0}^{N-1} x_n^2 \right) \tag{2.9}$$
$$:= \text{Mean-Square Leistung des Signals}$$

$$P_{dc} = \left(\frac{1}{N} \cdot \sum_{n=0}^{N-1} x_n \right)^2 \tag{2.10}$$
$$:= \text{Gleichleistung des Signals}$$

Man kann deutlich erkennen, dass insgesamt nur drei Leistungen bestimmt werden müssen, um den Indikator für das HDI Verzerrungsmaß zu berechnen. Bei der Bestimmung des HDI Verzerrungsmaßes wirkt sich durch diese Art der Berechnung die Rauschleistung auf den Indikator direkt aus. Die vollständige Rauschleistung geht unmittelbar in die Gesamtwechselleistung des Signals ein. Folglich wird dieses Verzerrungsmaß zum einen Teil durch harmonische Verzerrungen und zum anderen Teil durch die Verzerrungen, die durch höher frequentes Rauschen enstehen, beeinflusst.

2.1.2. Harmonic Distortion K (HD_K)

Als zweiten Indikator kann man eine Teilmenge der Werte, die in der HDI Berechnung genutzt werden, herausgreifen, um eine bandbegrenzte Untersuchung der harmonischen Verzerrungen durchzuführen. Durch diese Begrenzung übt das höher frequente Rauschen kaum noch Einfluss auf dieses neue Verzerrungsmaß aus. Eine differenziertere Betrachtung des Arbeitspunktes ist damit gewährleistet.

Dieses Verfahren wird im Projekt als HD_K geführt und ist an die von Temme in [33] definierten *Total Harmonic Distortion* (% THD) angelehnt.

Die Variable K steht hierbei für die Anzahl der Harmonischen (inklusive der Grundwelle), die für die Berechnung der Verzerrungen miteinbezogen wird. Die Formel für den HD_K wurde wie folgt definiert:

$$\mathrm{HD}_{K,\%} = 100 \cdot \sqrt{\frac{\sum_{i=2}^{K} |H_i|^2}{\sum_{i=1}^{K} |H_i|^2}} \ \% \tag{2.11}$$

Für dieses Verfahren müssen die K-Leistungsspektrallinien des Signals bestimmt werden. Zur Berechnung dieses Indikators war es bisher nicht möglich ein effizienteres Verfahren einzusetzen (im Gegensatz zum *HDI*), dies war die Motivation in der vorliegenden Arbeit genauer zu untersuchen, ob es möglich sei ein effektiveres Verfahren zu finden.

Man kann unter Verwendung von Parsevals Theorem zeigen, dass eine HD_K Berechung mit $K = N_{DFT}$ mit der Berechnung des HDI übereinstimmt. Die Berechnung beider Indikatoren für das Verzerrungsmaß ist aus diesem Grunde nur sinnvoll für $K << N_{DFT}$. Da man zwei verschiedene Indikatoren erhalten möchte, die einen unterschiedlichen Analysefokus besitzen.

Mit diesem Indikator ist es sehr gut möglich eine bandbegrenzte Analyse der harmonischen Verzerrungen des Signals durchzuführen, die das hochfrequente Rauschen unberücksichtigt lässt.

Für die Berechnung des HD_K kann man in der Praxis eine spektral reduzierte DFT verwenden, um die wenigen Spektrallinien die benötigt werden, zu bestimmen. Man muss jedoch genau untersuchen, für welche Werte von K sich eine Realisierung mittels der Fast-Fourier Transformation (FFT) lohnt. Wie bereits erwähnt, gibt es bisher in diesem Projekt keine Möglichkeit diese Berechnung deutlich effizienter zu realisieren. Zur Steigerung der Effizienz wird in dieser Masterarbeit ein neues Verfahren zur Spektralschätzung eingeführt.

2.2. Ordnungsanalyse

Im Rahmen der Analyse von ABS-Sensorsignaldaten wird im Projekt ESZ-ABS die sogenannte Ordnungsanalyse eingesetzt. Ein Zeitsignal wird unter Verwendung von Abtastung in die Winkeldomäne überführt. Dies ermöglicht eine differenziertere Analyse des Nutzsignals in Bezug auf die harmonischen Frequenzanteile.

Zur Analyse drehender Maschinen wird die Ordnungsanalyse häufig angewendet, da die harmonischen Verzerrungen genau analysiert werden können. Auf unsere Anwendung bezogen, bedeutet dies, dass die drehende Maschine, die überwacht wird, das Encoderrad auf der Radnabe darstellt. Durch die winkelsynchrone Abtastung wird nun zuerst ein sogenanntes Winkelsignal erzeugt, welches anschließend mittels Fourier-Transformation in den Bildbereich überführt wird. Man erhält das sogenannte Ordnungsspektrum.
Dieses Verfahren ähnelt der diskreten Fourier-Analyse sehr, unterscheidet sich jedoch darin, dass z.B. die Beschleunigungskomponente des Nutzsignals nicht erfasst wird, die Abtastzeitpunkte entsprechend variiert werden (siehe Abbildung 2.1). So erhält man ein periodisches Signal, das z.B. sehr genau auf seine Oberwellen untersucht werden kann.

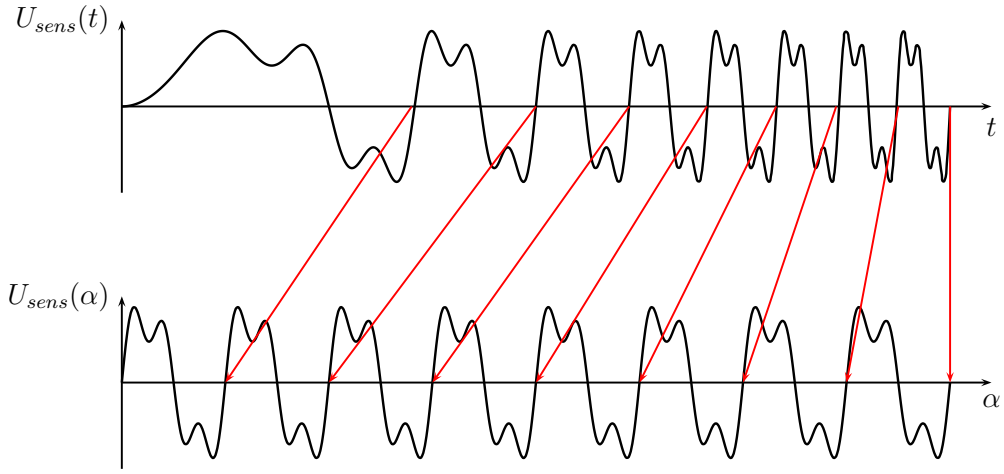


Abbildung 2.1.: Der Übergang des Zeitsignals mittels winkelsynchroner Abtastung in den Winkelbereich ermöglicht es auch im Beschleunigungsfall in der Winkeldomäne ein periodisches Signal zu erzeugen (Graphik entnommen aus [16]).

2.2.1. Prädiktion der Periodendauer

Die unbekannte Drehfrequenz des Encoderrades erfordert es, dass eine Schätzung der Periodendauer vorgenommen wird. In der Annahme, dass es sich um ein sinusförmiges Signal handelt, wird im Signal nach Nulldurchgängen gesucht, um diese als Referenz für die winkelsynchrone Abtastung zu nutzen. Im Forschungsprojekt wurde sich bewusst dafür entschieden die Nulldurchgänge des Signals als Referenz zu verwenden, da diese bei sinusförmigen Signalen sehr gut genutzt werden können, um auf eben diese Periodendauer zu schließen (und damit auch auf die Drehfrequenz des Rades).
Das Eingangssignal wird mit den Daten die durch die Nulldurchgangserkennung entstanden sind mit einer entsprechenden Rechteckfensterfunktion gewichtet, um genau eine Periode des Nutzsignals herauszugreifen. Hierfür wird angenommen, dass die Beschleunigung des Encoderrades so gering ist, dass die Nulldurchgänge der vorhergehenden Periode ausreichend sind, um die der nächsten Signalperiode hinreichend genau zu bestimmen. Aus dieser Erfassung wird im darauf folgenden Schritt die nächste Schätzung der Periodendauer berechnet.

2.2.2. Abtastung einer Zahnperiode

Ein erster Ansatz beruhte darauf, genau eine Signalperiode des Nutzsignals aus dem Signal zu extrahieren und diese Periode anschließend spektral zu analysieren. Typischerweise liefert der Sensor bei linearem Übertragungsverhalten exakt eine Periode des zu analysierenden Signals beim Übergang von einem Zahn des Encoderrades zum nächsten.

In der Praxis werden für diesen Zweck insgesamt drei Nulldurchgänge im Nutzssignal erwartet, da eine Periode eines mittelwertfreien Sinus genau diese drei Nulldurchgänge ausprägt.

Die Prädiktion der Periodendauer wird dementsprechend konfiguriert und die Abtastlogik wird so eingestellt, dass die Anzahl der Abtastwerte pro Periode genau in dieses Zeitfenster passt. Das Sensorsignal kann nun äquidistant abgetastet und verarbeitet werden.

Verwendet man nun dieses Bezugssystem als Ausgangspunkt für die diskrete Fourier-Transformation, erhält man das sogenannte Ordnungsspektrum, welches direkt die harmonischen Anteile des Signals als Indizes enthält (siehe Abbildung 2.2).

Nachteilig stellte sich jedoch heraus, dass Sensorsignale, die sehr ausgeprägte nichtlineare Verzerrungen aufweisen, bedingt durch einen abweichenden Arbeitspunkt des Sensors, nicht korrekt analysiert werden können (in Bezug auf ein Verzerrungsmaß), da eine Fehlinterpretation der Grundperiode erfolgt. Im Spektrum des Signals kann sich diese Fehlinterpretation z.B. als falscher Gleichanteil erweisen.

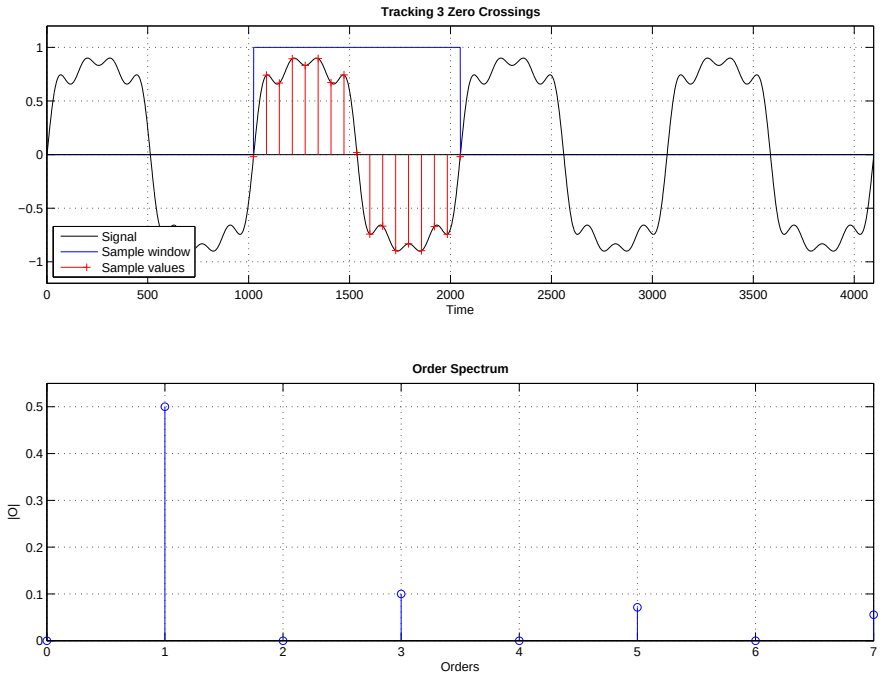


Abbildung 2.2.: Die Abtastung mithilfe von drei Nullstellen als Referenzfenster liefert ein Ordnungsspektrum, in dem die Ordnungen direkt mit den Harmonischen Anteilen übereinstimmen (für Signale deren Grundwelle durch diese drei Nullstellen erfasst werden kann).

Dieses Verhalten wird im Forschungsprojekt ESZ-ABS als Frequenzverdopplung bezeichnet. Ursächlich hierfür ist die Tatsache das bei dem Übergang von einer Zahnperiode zur nächsten ein Signal mit zwei Perioden entsteht. Die zweite Harmonische des Signals wird nämlich viel zu stark ausgeprägt (siehe dafür Abbildung 2.3), sodass die Nullstellen nicht mehr als Referenz für die Periodendauer der Grundwelle genutzt werden können.

Im Rahmen des Projektes zeigte sich, dass eine solche Frequenzverdopplung auf Grund des nichtlinearen Kennlinienfeldes des Sensors durchaus vorkommen kann, weshalb diesem Thema erhöhte Beachtung geschenkt werden muss.

Zur Lösung dieses Problems wurde die sogenannte Zweizahnperiodenabtastung eingeführt, auf die im folgenden Abschnitt eingegangen wird.

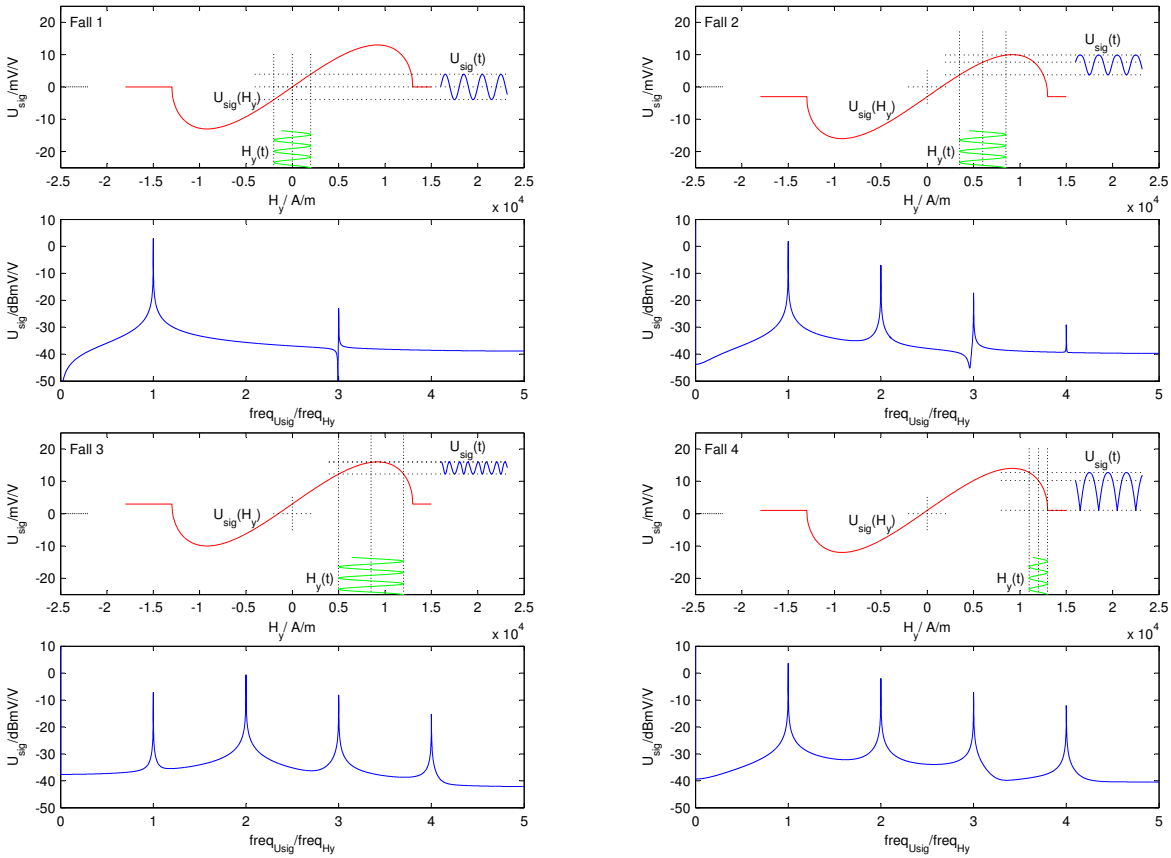


Abbildung 2.3.: Die oberen vier Teilbilder zeigen die magnetische Anregung des Sensors einmal im nahezu linearen (oben links) sowie in einem nur leicht nichtlinearen Bereich der Kennlinie. Dazugehörig wird das Spektrum des Sensorsignals angegeben. Das Zustandekommen der sogenannten Frequenzverdopplung wird in den unteren vier Teilbildern sehr gut deutlich. Durch die sinusförmige Anregung der Sensorenkennlinie in den nichtlinearen Bereichen (unten links und unten rechts) entsteht im Spektrum des Sensorsignals eine sehr ausgeprägte 2. Harmonische (Abbildung modifiziert und entnommen aus [24]).

2.2.3. Abtastung von zwei Zahnperioden

Für die Zweizahnperiodenabtastung werden insgesamt fünf Nulldurchgänge des zu analysierenden Signals als Referenz gewählt. Man versucht also den Übergang des Sensors über zwei Zähne des Encoderrades zu erfassen.

Im Idealfall erhält man dadurch innerhalb dieses Zeitfensters zwei Perioden des Nutzsignals. Liegt keine Frequenzverdopplung vor, so handelt es sich bei diesen beiden Perioden tatsächlich um zwei Perioden des Nutzsignals (siehe Abbildung 2.4).
Sofern jedoch eine Frequenzverdopplung auf Grund des nichtlinearen Kennlinienfeldes vorliegt, erhalten wir zwei Pseudoperioden des zugrunde liegenden Signals. Da die eigentliche Grundwelle kleiner ist als die der doppelten Frequenz, erhält man ein sehr stark verzerrtes Signal. Führte man die Zeit-Winkel Zuordnung anhand von nur drei Nullstellen bei einem Signal mit Frequenzverdopplung durch, erhielte man nicht das korrekte Ordnungsspektrum (siehe Abbildung 2.5).

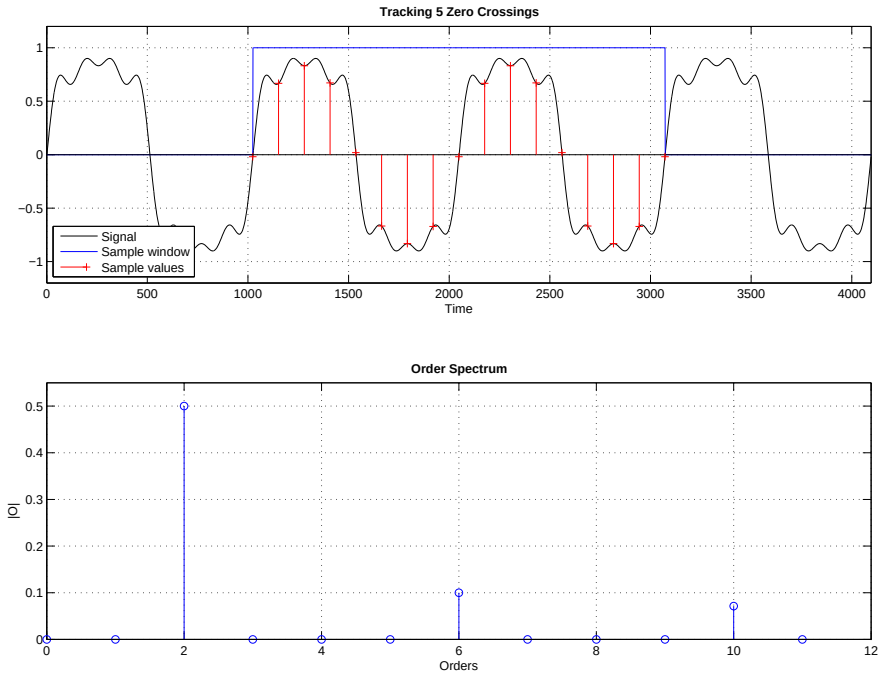


Abbildung 2.4.: Diese Abbildung demonstriert exemplarisch die Abtastung von zwei Zahnperioden, eines Signals ohne Frequenzverdopplung. Es ist deutlich zu sehen, dass sich durch diese Art der Winkelumsetzung das Ordnungsspektrum verändert (in Bezug auf die Abtastung von einer Zahnperiode), da die Grundwelle nun an Index 2 zu finden ist.

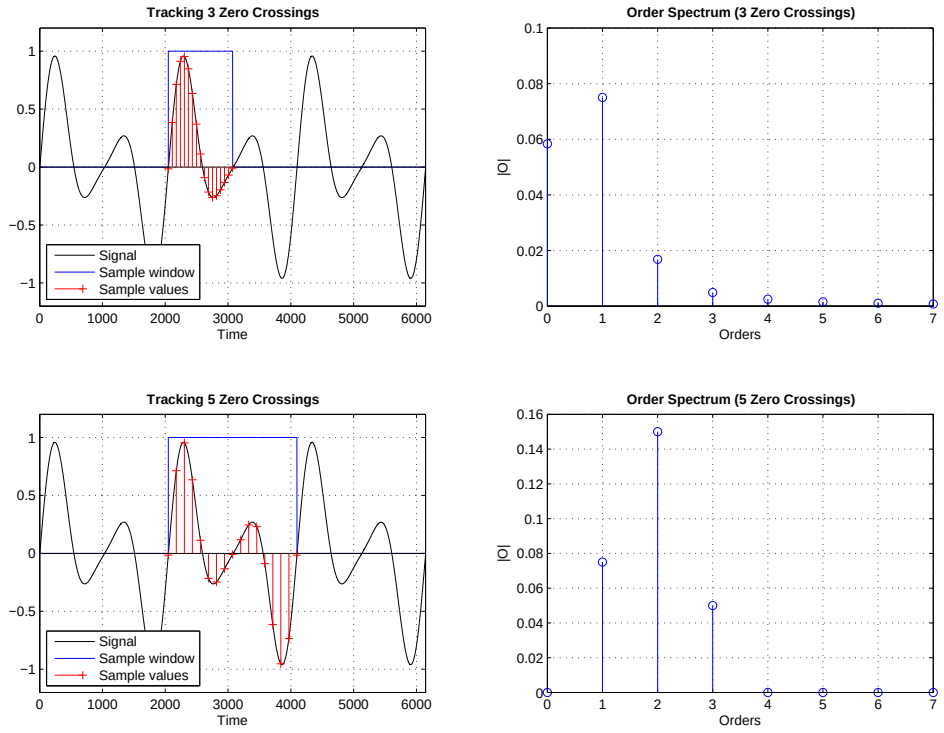


Abbildung 2.5.: Die Detektion von Signalen mit Frequenzverdopplung ist nur möglich, wenn die Abtastung mit fünf Nullstellen als Referenzfenster durchgeführt wird. Im oberen Teil des Bildes ist es ersichtlich, dass die Frequenzverdopplung nicht korrekt erkannt werden kann, da sich diese lediglich als falscher Gleichanteil im Signal bemerkbar macht. Im unteren Teil der Abbildung ist deutlich zu erkennen, dass nun ein zusätzlicher Anteil im Ordnungsspektrum an Index 1 zu finden ist. Dieser lässt auf eine darunterliegende Grundwelle schliessen und ermöglicht damit die Erkennung der Frequenzverdopplung.

Durch die Änderung dieser Winkelabtastung liefert die diskrete Fourier-Transformation keine Eindeutigkeit mehr in Bezug auf die Interpretation des Signals anhand der Indizes. Es muss immer eine Fallunterscheidung getroffen werden, um auf die Bedeutung der Indizes zu schließen.

1. Liegt die Frequenzverdopplung vor, so handelt es sich bei dem ersten Index um den spektralen Anteil der Grundperiode.
2. Ist dies jedoch nicht der Fall, so liefert der zweite Index den spektralen Anteil der Grundwelle.

Die Basis für diese Art der Abtastung wurde im Forschungsprojekt ESZ-ABS von Poppinga [23] eingeführt. Im Rahmen seiner Bachelorthesis wurden noch einige Lösungskonzepte für die Erkennung der Frequenzvordopplung mittels Even Harmonic Distortion (EHD) und Odd Harmonic Distortion (OHD) vorgeschlagen. In dieser Masterarbeit finden sie jedoch keine Verwendung, da die Berechnung der Standardverzerrungsmaße ausreichend ist.

2.3. Reduktion der DFT auf wenige Spektrallinien (rDFT)

Die diskrete Fourier-Transformation (DFT) stellt das elementare Mittel zur Spektralanalyse zeitdiskreter Signale in der heutigen Signalverarbeitung dar. Mit ihr ist es relativ einfach möglich, Signale durch harmonische Funktionen (Sinus und Cosinus) auszudrücken. Die DFT wird als die Fourier-Analyse für endliche zeitdiskrete Signale bezeichnet, da eine endliche Folge von Abtastwerten in eine endliche Folge von komplexen Sinusoiden transformiert wird. Die Elemente dieser Folge repräsentieren die komplexen Frequenzkomponenten des Signals und ermöglichen die Betrachtung des Spektrums des diskreten Signals.

In den bisherigen Arbeiten wurde eine sogenannte reduzierte DFT (rDFT) verwendet, weil im Vergleich zur theoretisch möglichen Bandbreite der DFT nur wenige harmonische Frequenzkomponenten des Signals analysiert werden müssen. Die reduzierte DFT Implementierung berechnet nur die spektralen Anteile des Signals, die tatsächlich genutzt werden, um auf ein Verzerrungsmaß schliessen zu können.

Beispielsweise liefert eine 64-Punkte DFT eines einperiodisch abgetasteten winkelsynchronen Signals die Möglichkeit 32 verschiedene harmonische Spektralkomponenten zu erhalten. Auf Grund der spektralen Symmetrie reeller Eingangsfolgen (siehe dafür Abbildung 2.7) stimmen die übrigen 32 Spektralkomponenten abgesehen von einem Spiegelfaktor mit den vorherigen überein [1]. Von diesen 32 Harmonischen werden jedoch in den bisherigen Realisierungen nur fünf Harmonische ausgewertet. Folglich werden unter 20% der Anteile, die bei einer vollständigen 64-Punkte DFT berechnet werden würden, tatsächlich genutzt.

In der Arbeit von Dreschhoff (siehe [3]) wurde gezeigt, dass genau aus diesem Grund die Berechnung einer rDFT in Bezug auf die Hardwareressourcen deutlich angemessener für ein minimales System ist. Es müssen keinerlei Informationen berechnet bzw. gespeichert werden, die nicht unmittelbar in das Verzerrungsmaß eingehen.

Bei der reduzierten DFT werden lediglich die spektralen Komponenten des Signals berechnet, die hauptverantwortlich für die Verzerrungen sind. Unter Verwendung dieser Spektrallinien wird anschließend das HD_K Verzerrungsmaß berechnet. Dies hat zur Folge, dass lediglich K-Spektrallinien tatsächlich berechnet werden müssen.

Nehmen wir beispielshalber an, dass das Nutzsignal spektral dünn besetzt ist und demzufolge nicht mehr als vier Oberwellen besitzt. Unter dieser Annahme genügt es nun diese vier Oberwellen und die Grundwelle des Signals spektral zu analysieren und in ein Verzerrungsmaß (HD_5) zu überführen.

Auch in der Praxis zeigte sich, dass dies für die Signale, welche von ABS-Sensoren erzeugt werden, durchaus zutrifft. Ist die Spitzenspannung des Signals groß genug und damit das Rauschen vernachlässigbar klein, so ist es ausreichend, lediglich fünf Harmonische zu untersuchen. In der Abbildung 2.6 wird ein solches Signal exemplarisch dargestellt.

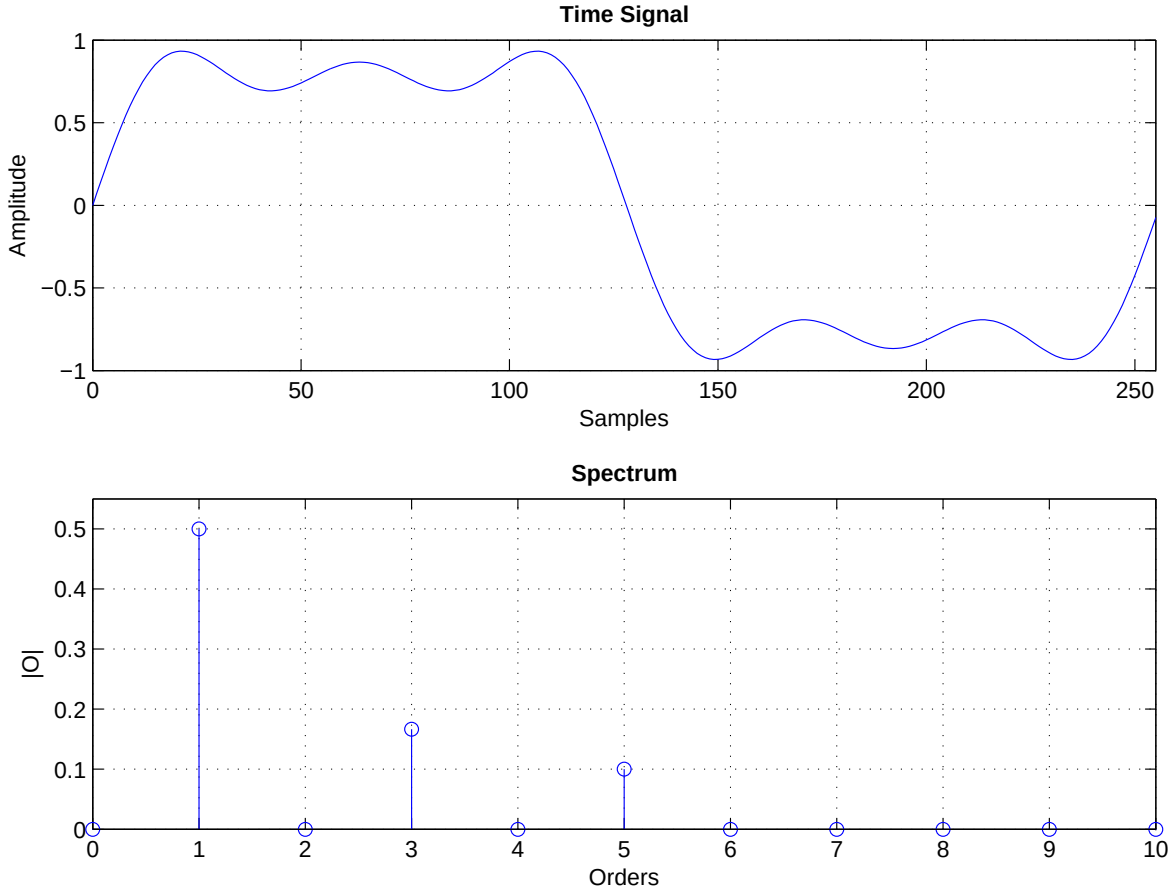


Abbildung 2.6.: Die Darstellung eines typischen Signals wie es im Einsatz von AMR basierten ABS-Sensoren vorkommt. Die Verzerrungen kann man im Ordnungsspektrum an den Indizes 3 und 5 erkennen. Auf Grund des nichtlinearen Kennlinienfeldes des AMR-Sensors sind sie zusätzlich zur Grundschwingung (ersichtlich an Index 1) vorhanden.

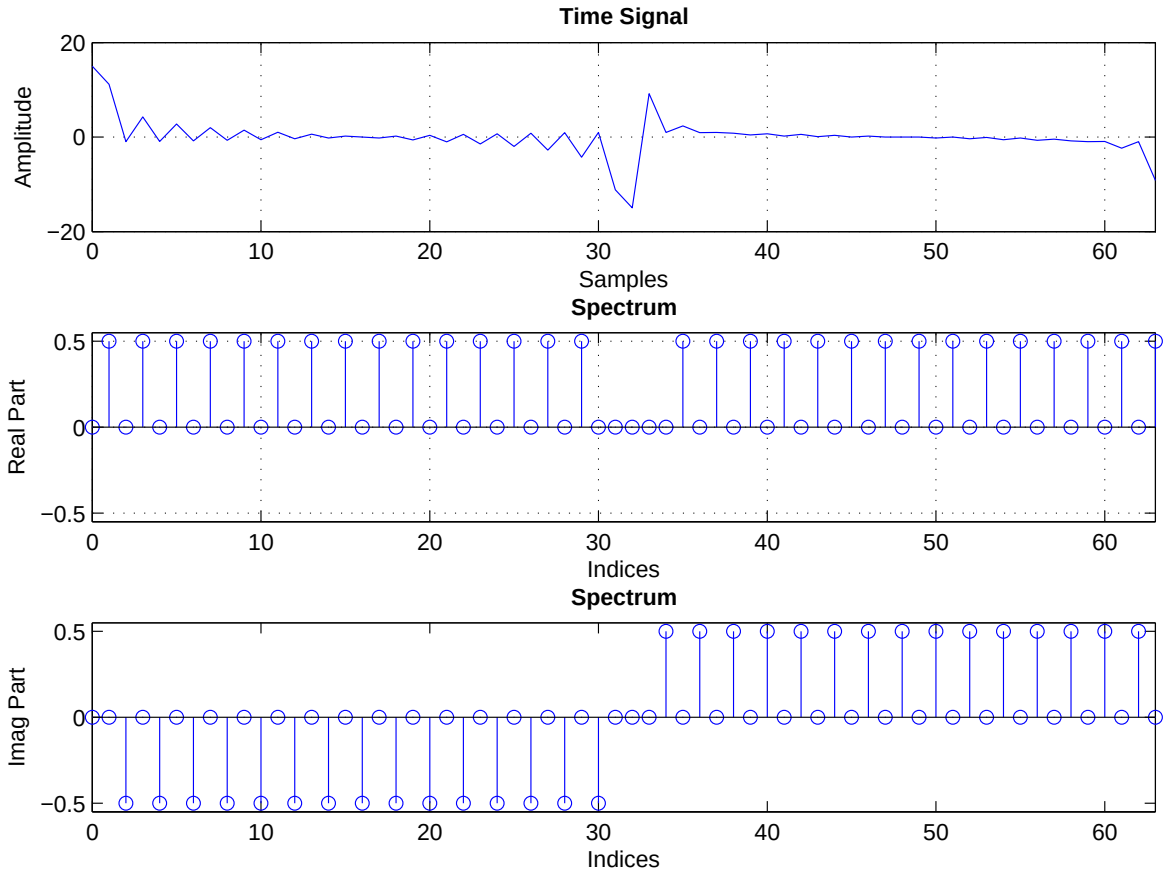


Abbildung 2.7.: Diese Abbildung demonstriert die Symmetrie der DFT für reelle Eingangssignale x_n. Beispielhaft wird ein Signal mit 64 Samples dargestellt. Der Realteil verhält sich immer spiegelsymmetrisch, während der Imaginärteil eine Punktsymmetrie aufweist.

3. Analyse

3.1. Stand der bisherigen Arbeiten im Projekt ESZ-ABS

Im Projekt ESZ-ABS wurden bisher viele Bachelor-, Master- sowie Diplomarbeiten durchgeführt, die sich den verschiedensten Aspekte der Zustandserkennung bzw. den genauren Analysen der auf dem AMR-Effekt basierenden ABS-Sensoren widmeten.

Das Projekt lässt sich dabei grob in drei Kategorien aufteilen. Der erste Aspekt hierbei ist die Entwicklung von Kreuzspulenmessplätzen, damit die Kennlinie bzw. das zweidimensionale Kennlinienfeld aufgenommen werden kann und um ein besseres Verständnis über das Verhalten von diesen speziellen ABS-Sensoren zu erhalten.

Desweiteren wurden verschiedene Radmessplätze entwickelt und aufgebaut, um Messdaten unter Verwendung von verschiedenen Sensoren und Encoderrädern aufzunehmen, die dann als Basis für die verschiedenen Signalverarbeitungsalgorithmen genutzt werden können.

Zur Umsetzung der gewonnene Erfahrungen aus den Messplätzen wurden zudem diverse Experimentalplattformen entwickelt. Diese ermöglichen die praktische Anwendung von Signalverarbeitungskonzepten. Angefangen von Jegenhorst [9] wurde 2009 eine auf dem TI-MSP430 Mikrocontroller basierende erste Experimentalplattform entworfen, die es erstmals ermöglichte, die digitale Signalverarbeitung auf einem technischen System hardwarenah durchzuführen. Ausgehend von dieser Plattform wurden diverse weitere Verbesserungen des Systems mit neuen analogen Frontends sowie anderen Ansätzen zur Signalverarbeitung entwickelt.

Im Jahr 2010 wurde von Dreschoff [3] ein FPGA-Prototyp für die Signalverarbeitung von ABS-Sensoren geschaffen. Dieser Prototyp stellte die Basis für die erste Testchip-Implementierung der Signalverarbeitungsalgorithmen dar, die von Sabotta [27] in 2012/13 durchgeführt wurde.

Nach dieser Implementierung stellte sich die Frage, ob es möglich sei, den Flächenbedarf für die Signalverarbeitung weiter zu senken. Die Vision der ABS-Sensoren mit Selbstdiagnosefunktionalität erfordert es, dass die implementierten Algorithmen in digitaler Hardware und das analoge Frontend des Diagnose Systems möglichst klein und stromeffizient

arbeiten müssen. Hierbei handelt es sich einerseits um ein Massenprodukt, welches millionenfach im Jahr gefertigt wird, andererseits bestehen sehr strikte Auflagen, die z.B. die Stromaufnahme und das Kommunikationsprotokoll dieser Sensoren eingrenzen.

In Abbildung 3.1 wird dargestellt, welche Vorarbeiten einen signifikanten Einfluss auf diese Masterarbeit hatten. Dabei ist zu beachten, dass es sich hierbei lediglich um einen Ausschnitt des gesamten Projektes handelt. Das Projekt umfasst mittlerweile 18 Arbeiten, die natürlich nicht zu vernachlässigen sind, da auch diese einen sehr großen Einfluss auf das Gesamtprojekt hatten.

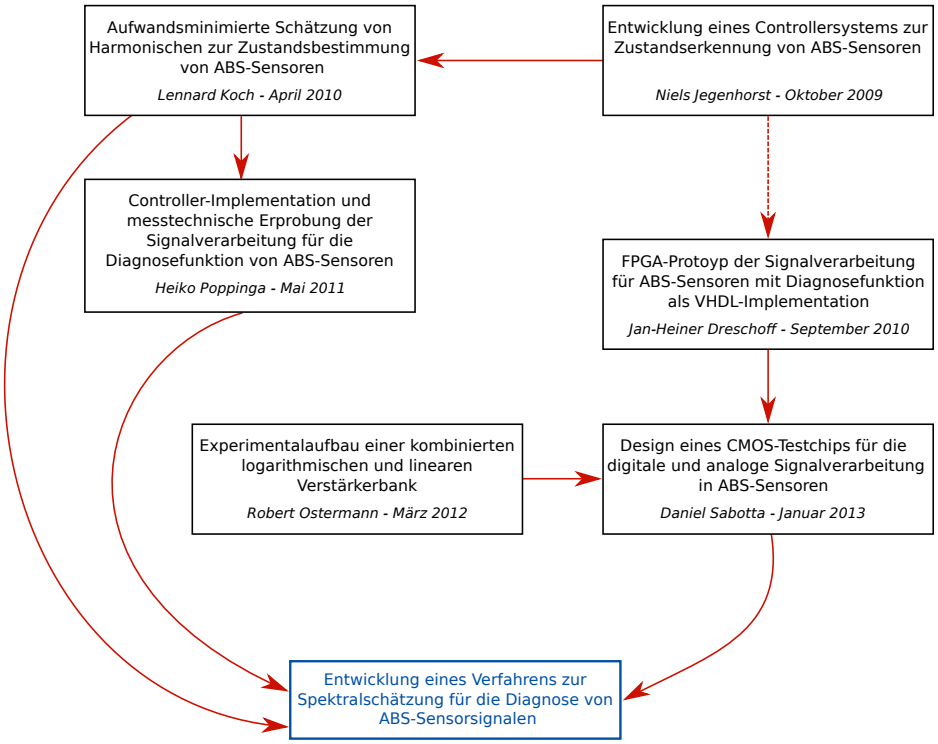


Abbildung 3.1.: Ausschnitt aus den bisherigen Abschlussarbeiten im Projekt ESZ-ABS mit Fokus auf den für diese Masterthesis relevanten Vorarbeiten. Der linke Zweig repräsentiert die Entwicklung der Mikrocontroller-Experimentalplattform, während der rechte Zweig die Entwicklung der FPGA/CMOS-Experimentalplattformen darstellt.

3.2. Untersuchung der Testchip-Implementierung

Für eine genauere Betrachtung der bisherigen Chip-Implementierung (siehe Abbildung 3.4) wird diese im Folgenden in Bezug auf die benötigte Chip-Fläche analysiert. Unter Verwendung des Cadence Encounter RTL Compilers (Cadence ERC) wurde die Flächenbelegung der einzelnen Verarbeitungsdomänen untersucht. Dies geschah in Zusammenarbeit mit Sabotta [27]. Hierbei ist zu beachten, dass nur der Flächenbedarf der Standardzellen durch den Synthese-Compiler ausgewertet wird. Die eventuell vorhandenen Freizellen, die vom Autorouter genutzt werden, um diese Standardzellen miteinander zu verbinden, sind in dieser Kalkulation ebenfalls nicht mit eingeschlossen.

Die bisherige Implementierung lässt sich in fünf Domänen unterteilen:

- die Abtastlogik (Sample Logic),
- die Periodendauerabschätzung (Period Estimation),
- die Diskrete Fourier-Transformation (DFT),
- das sogennante Postprocessing (Postprocessing),
- und zuletzt die Logik, die verwendet wird, um die verschiedenen Domänen miteinander zu verbinden (Glue Logic).

In Abbildung 3.2 wird diese Flächenbelegung aufgezeigt. Dabei wird es sehr deutlich, dass das Postprocessing und damit die Berechnung der Verzerrungsmaße, die größte Fläche einnimmt. Mit ca. 23% folgt die Berechnung der reduzierten DFT. Die anderen Domänen sind vernachlässigbar klein und bieten nicht sehr viel Handlungsraum für weitere Optimierungen.

Diese strikte Trennung der Domänen ist in der Praxis nicht direkt umsetzbar, da es bei der Hardware Entwicklung oftmals üblich oder gar nötigt ist, die Domänen miteinander zu vermischen. Zudem gibt es auch einige Abhängigkeiten zwischen den Domänen, welche sich auch auf den Flächenbedarf der nachfolgenden Module stark auswirken.

Die rDFT zum Beispiel benötigt eine definierte Auflösung der Lookup-Tabellen Werte für die verwendeten Sinus- bzw. Cosinus-Funktionen (z.B. 10-Bit). Ändert man nun diese Auflösung, so wirkt sich dies direkt auf den Flächenbedarf der rDFT aus, da der benötigte Multiplizierer angepasst werden muss.

Außerdem ist es oft üblich in Hardware Systemen mit erweiterten Bitbreiten fortzufahren, bis diese in den letzten Schritten der Verarbeitung herunterskaliert werden können. Folglich hängt somit auch der Flächenbedarf des Postprocessings von der Auflösung der rDFT ab.

Zur weiteren Untersuchung wurde erneut unter Verwendung des Cadence ERCs eine Analyse durchgeführt, welche den Flächenbedarf der einzelnen Logiktypen kategorisiert.

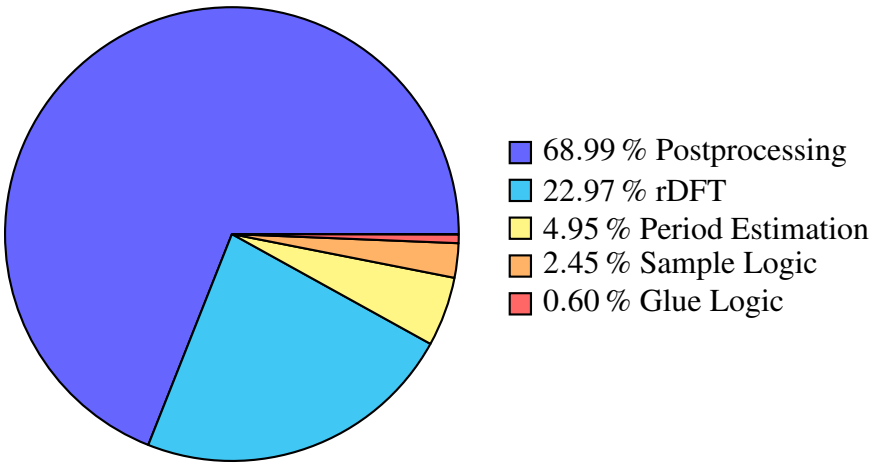


Abbildung 3.2.: Prozentualer Flächenbedarf der bisherigen Chip-Implementierung in Bezug auf die Signalverarbeitungsdomänen

Hierbei kann wiederum zwischen drei Typen unterschieden werden:

- die kombinatorische Logik (Standard Gatter z.B. NAND, XOR, MUX, ...),
- die sequentielle Logik (getaktete Gatter mit speichernder Funktion z.B. D-Flip-Flops),
- und einfache Inverter die zum Teil auch als Buffer genutzt werden.

In Abbildung 3.3 wird diese Analyse veranschaulicht. Sie zeigt, dass die sequentielle Logik absolut gesehen den größten Anteil der Fläche einnimmt. Dies ist jedoch darauf zurückzuführen, dass ein großer Anteil dieser Ressourcen für Testschnittstellen und zur Speicherung von Zwischenergebnissen für Testzwecke genutzt wird. In einer finalen Implementierung ist hier noch großes Optimierungspotential vorhanden.

Bei genauer Betrachtung fällt sofort auf, dass die Instanzen der sequentiellen Logik einen wesentlich höheren Flächenbedarf pro Standardzelle haben. Im Durchschnitt nimmt eine Instanz sequentieller Logik $\approx 0{,}03\%$ der gesamten Fläche ein, während eine kombinatorische Standardlogikzelle durchschnittlich $\approx 0{,}01\%$ belegt. Verringert man demnach die Anzahl der Register, so spart man sehr effizient Fläche ein. Die Einsparung eines Flip-Flops ist dementsprechend mit der von durchschnittlich drei Standardlogikgattern gleichzusetzen. Dies ermöglichte einen ersten Ansatz zur Entwicklung einer Optimierungsstrategie.

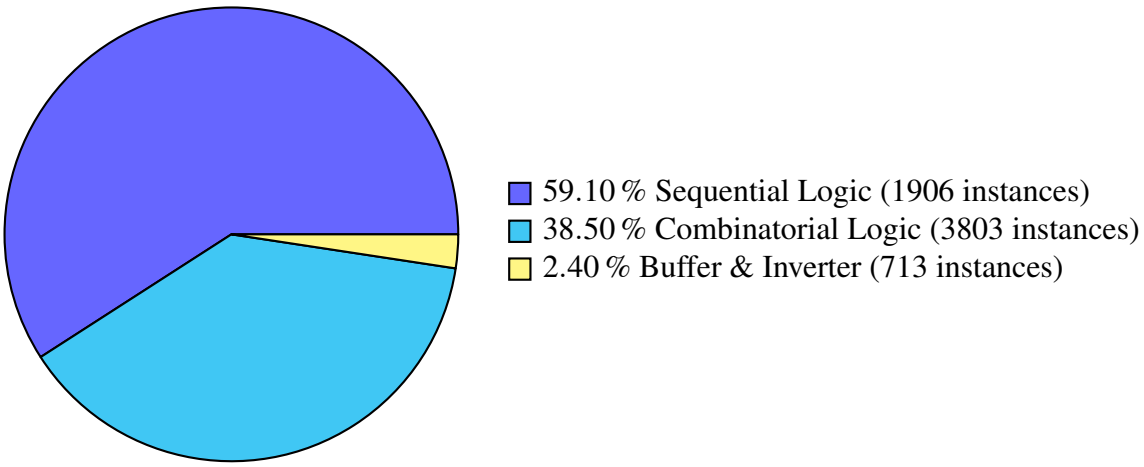


Abbildung 3.3.: Prozentualer Flächenbedarf der Implementierung der Diagnostik-Einheit in Bezug auf die verschiedenen genutzten Logiktypen

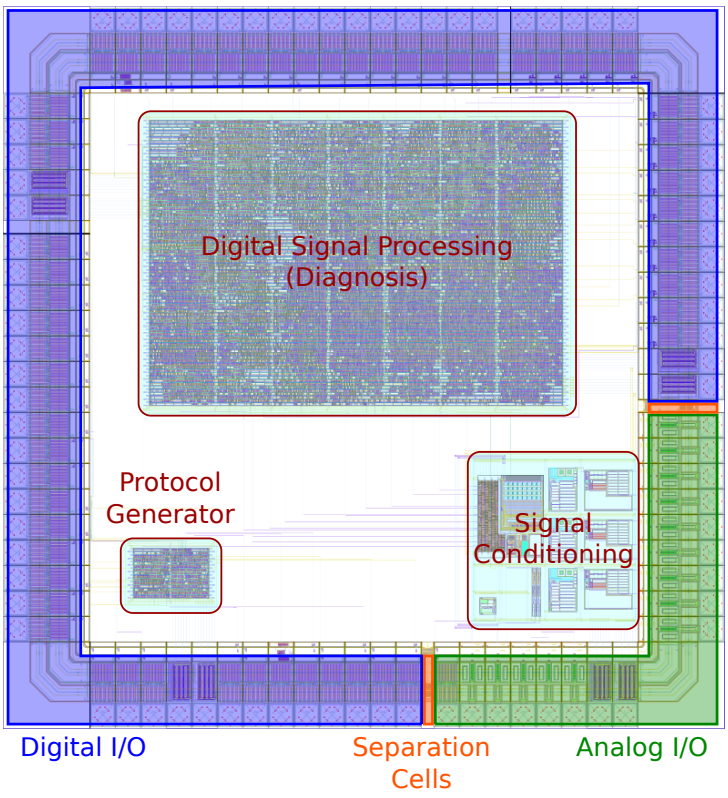


Abbildung 3.4.: Darstellung des Testchip-Layouts mit Kennzeichnung der einzelnen Domänen (entnommen aus [27] und modifiziert).

3.3. Analyse der gewählten Indikatoren für das Verzerrungsmaß

Im Grundlagenkapitel wurde bereits erwähnt, dass mehrere Indikatoren für das Verzerrungsmaß existieren. Diese beiden Indikatoren geben jedoch unterschiedlichen Aufschluss über den Zustand des Sensors.

In der bisherigen Implementierung werden beide Indikatoren berechnet. Eine tatsächliche Notwendigkeit zur Nutzung beider Indikatoren konnte bisher nicht geradlinig bestimmt werden. Sie müssen differenziert betrachtet werden, da sie jeweils Vor- und Nachteile besitzen.

Der HD_K Indikator z.B. liefert ein sehr gutes Maß für die harmonischen Verzerrungen in einem stark eingeschränkten Bereich, da nur K-Harmonische einbezogen werden. Dies ermöglicht es eine sehr gute Aussage über die nichtlinearen Verzerrungen zu treffen, die durch das Kennlinienfeld des ABS-Sensors gegeben sind. In realen Systemen ist es jedoch häufig der Fall, dass ein hochfrequentes Rauschen dem Nutzsignal additiv überlagert ist. In der Berechnung des HD_K findet dies keinerlei Beachtung.

Demzufolge wird der HDI als zweiter Indikator zusätzlich zum HD_K verwendet, da dieser die Grundwelle zur Gesamtwechselleistung des Nutzssignals bewertet. Ein vorhandenes hochfrequentes Rauschen kann sich aus diesem Grund auf das HDI-Verzerrungsmaß direkt auswirken.

In Abbildung 3.5 ist dargestellt wie sich die beiden Indikatoren über das sogenannte Airgap, also den Abstand zwischen Encoderrad und Sensor bei idealem Einbau, verhalten. Durch die Verwendung beider Verzerrungsmaße in Kombination kann man besser auf den Arbeitspunkt des Sensors schließen, als es mit nur einem Indikator möglich wäre.

Durch die empirische Analyse von Krey [16] konnte gezeigt werden, dass die Verwendung von fünf Spektrallinien für das HD_K-Verzerrungsmaß ausreichend ist. Auf Grund dessen wird in den meisten bisherigen Realisierungen (siehe Koch [12] und Jegenhorst [9]) das HD_5-Verzerrungsmaß verwendet. Auch in dieser Masterarbeit wird demzufolge der HD_5 eingesetzt.

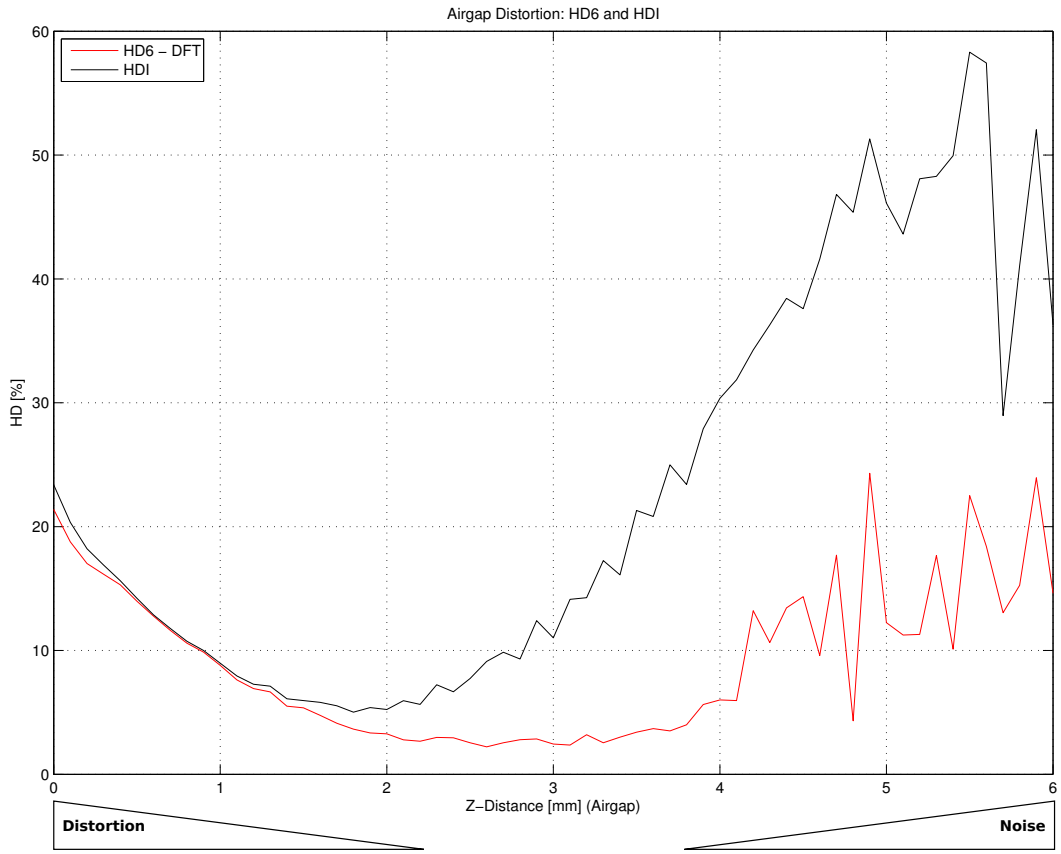


Abbildung 3.5.: Darstellung der beiden Indikatoren des Verzerrungsmaßes über das Airgap zwischen ABS-Sensor und Encoderrad.

3.4. Abgrenzung gegenüber anderen Verfahren

Zur Berechnung bzw. Schätzung spektraler Anteile eines Signals existieren bereits viele verschiedene Verfahren. Das meist verbreitete Verfahren bildet die Fast-Fourier-Transformation, die eine optimierte Version der diskreten Fourier-Transformation darstellt. Ebenso existiert jedoch noch die reduzierte DFT, welche im vorigen Kapitel bereits ausführlich besprochen wurde, sowie einige weitere Verfahren.

Zum Beispiel ist ein sehr bekanntes Verfahren, das oftmals zur Erkennung von DTMF-Tönen eingesetzt wird, der Goertzel-Algorithmus. Zur relativ einfachen Berechnung von einzelnen Spektrallinien werden Infinite-Impulse-Response Filter (IIR-Filter) 2. Ordnung eingesetzt (siehe [5]).

Die Abgrenzung gegenüber diesen Verfahren erfordert eine Untersuchung der Kriterien, die für eine ASIC-Realisierung von Bedeutung sind.

In der theoretischen Informatik wird z.B. oft die Anzahl von gewissen Operationen genutzt, um die Komplexität und das Laufzeitverhalten eines Algorithmus zu quantifizieren. Häufig wird für diesen Zweck die Multiplikation als Basisoperation gewählt. Für eine Hardwareimplementierung ist es jedoch oftmals von größerer Bedeutung ob eine Multipliziereinheit von mehrerern Ressourcen geteilt werden kann. Ein weiterer wichtiger Faktor ist ob die Datenverarbeitung on-the-fly durchgeführt werden kann. Dies bedeutet das ein Sample direkt verarbeitet wird ohne dieses selbst in einem Speicher bis zum Ablaufende des Algorithmus vorzuhalten.

Die Analyse des Testchips bzw. der Syntheseergebnisse zeigte, dass die Anzahl der Register einen sehr großen Einfluss auf die benötigte Chip-Fläche ausübt. Zielführend sollte es sein die Anzahl der Register für Resultate und Zwischenergebnisse möglichst gering zu halten (diese Abhängigkeit ist in Tabelle 3.1 einzusehen).
Außerdem ist auch die Bitbreite eines Registers eine nicht zu vernachlässigende Größe. Durch Skalierungsoperationen kann diese jedoch relativ gut angepasst werden und ist deshalb bei der Auswahl der Algorithmen vernachlässigbar. Zu beachten ist nur dass sich ein Verlust an Berechnungspräzision ergibt, wenn diese Skalierungen nicht im Nachhinein durchgeführt werden.

Die Anzahl der Multiplikationen ist natürlich auch nicht immer zu vernachlässigen, wenn z.B. ein sequentieller Multiplizierer eingesetzt wird. Die Abarbeitungszeit eines sequentiellen Multiplizierers skaliert linear mit der Bitbreite der Operanden, sodass auch die Abhängigkeit der Multiplikationen in Bezug auf den verwendeten Algorithmus in Tabelle 3.1 ersichtlich ist.

Wie bereits erwähnt ist es in diesem Projekt nötig eine on-the-fly Verarbeitung der Eingangsdaten durchzuführen, da kein zusätzlicher Speicher für das Vorhalten der Abtastwerte

genutzt werden soll. Diese Restriktion führt dazu, dass keine Algorithmen mit blockweisen Operationen genutzt werden können.

Algorithmus bzw. Verfahren	reelle Multiplikationen	Register
FFT	$N \cdot \log_2(N/4) + 2 \cdot N$	$2 \cdot N$
DFT	N^2	$2 \cdot N$
reduzierte DFT	$2(K \cdot N)$	$2 \cdot K$
Goertzel Algorithmus	$K(N+2)$	$4 \cdot K$

Tabelle 3.1.: Übersicht der benötigten reellen Multiplikationen und Register, die für die Anwendung der entsprechenden Algorithmen benötigt werden [10].

Algorithmus bzw. Verfahren	reelle Multiplikationen	Register
FFT	384	128
DFT	4096	128
reduzierte DFT	640	10
Goertzel Algorithmus	330	20

Tabelle 3.2.: Setzt man die in der bisherigen Chip-Implementierung verwendeten Parameter für die Evaluation der Tabelle 3.1 (Anzahl der Abtastwerte $N = 64$ und Anzahl der Harmonischen $K = 5$), so erhält man diese Tabelle als Resultat. Das führt zu einer besseren Betrachtung, da keine komplizierten Formeln ausgewertet werden müssen.

Die Fast-Fourier-Transformation (FFT)
Die schnelle Fourier-Transformation stellt die effizienteste Methode dar, wenn man alle N-Spektrallinien einer Folge der Länge N berechnen möchte (siehe [10]).
Sie ist eine spezielle Implementierung der diskreten Fourier-Transformation, welche auf Performanz optimiert wurde. Die Voraussetzung dafür ist es jedoch, dass die Länge der Eingangsfolge N eine Potenz von zwei ist. Es ist in der Praxis meist kein Problem, da die Anzahl der Abtastwerte oftmals frei gewählt werden kann.
In Mikroprozessorsystemen wird dieser Algorithmus häufig verwendet, da die Anzahl der Multiplikationen dort einen sehr viel höheren Stellenwert besitzt als der Speicherverbrauch eines Algorithmus. Der interne Speicher eines solchen Systems ist oft ausreichend vorhanden und die Zeitanforderungen an solche Systeme sind wesentlich höher, da diese mehr als nur einen Task verarbeiten müssen. Nachteilig sind hierbei der hohe Speicherbedarf sowie die Notwendigkeit zur Berechnung aller N-Spektrallinien.

Die diskrete Fourier-Transformation (DFT)
Aus den Tabellen 3.1 und 3.2 ersichtlich, ist die direkte Implementierung der diskreten Fourier-Transformation im Vergleich zu den anderen Verfahren die mit der geringsten Performanz. Es werden deutlich mehr Multiplikationen als bei den anderen Verfahren verwendet und zusätzlich wird auch noch sehr viel mehr Speicher genutzt, da alle Spektrallinien berechnet werden müssen. Aus diesem Grund ist der Einsatz dieser direkten Implementierung im Projekt ESZ-ABS nicht möglich.

Die reduzierte DFT
Im Grundlagenkapitel wurde die reduzierte DFT eingeführt. Die stellt momentan die Referenzimplementierung im Projekt dar. Das Verfahren erwies sich als sehr ausgewogen in Bezug auf Multiplikationen und Speicher und ist zusätzlich sehr einfach zu Implementieren. Die Verarbeitung von Abtastwerten kann on-the-fly geschehen und es kann ohne große Probleme nur ein Multiplizierer eingesetzt und zwischen den Ressourcen geteilt werden.

Der Goertzel-Algorithmus
Der Goertzel-Algorithmus wird in der Praxis angewandt, sofern nur einige wenige Spektrallinien berechnet werden müssen im Verhältnis zur Länge der Eingangsfolge.
Der Algorithmus wird als Filter mit unendlicher Impulsantwort (IIR-Filter) 2. Ordnung entworfen und implementiert. Es werden nur relativ wenige Multiplikationen benötigt, um eine Spektrallinie zu berechnen, und auch der Speicherbedarf ist relativ gering (siehe Tabelle 3.1). Es werden jedoch doppelt so viele Speicherzellen im Vergleich zu einer rDFT-Implementierung verwendet.

Hingegen sind IIR-Filter nicht immer stabil und können parasitäre nichtlineare Eigenschwingungen zeigen. Zudem ist die Filterrauschleistung eines IIR-Filters meistens größer als die eines FIR-Filters, also eines Filters mit endlicher Impulsantwort (siehe [34]).

Aus diesen Gründen wurde das Verfahren der reduzierten DFT bisher der Goertzel-Implementierung vorgezogen. In zukünftigen Arbeiten sollte erneut untersucht werden, ob es zweckmäßig sein könnte diesen Algorithmus zu verwenden.

Weitere Verfahren auf der Basis von Rechtecksignalen
Weiterhin existieren bereits einige Verfahren, die Rechtecksignale als Basis verwenden, um Transformationen vom Zeit- in den Bildbereich durchzuführen. Diese Verfahren werden häufig in der Bildverarbeitung eingesetzt und deren Resultate sind nicht immer direkt mit dem Spektrum der DFT vergleichbar.
Als Beispiel dafür sei die Hadamard-Transformation angeführt (auch bekannt als Walsh-Hadamard-, Hadamard-Rademacher-Walsh-, Walsh- oder Walsh-Fourier Transformation).
Für einige wenige Fälle lässt sich das Resultat dieser Transformationen direkt mit dem der DFT vergleichen. Zumeist jedoch sind weitere komplexe Transformationen notwendig, um ein Resultat in das andere zu überführen.
Überdies existieren auch weitere ganzzahlige Transformationen z.B. die arithmetische Fourier-Transformation (AFT) oder die grob quantisierte diskrete Fourier-Transformation

(QDFT). Diese benötigen eine zusätzliche Transformation, um die spektralen Anteile des Resultats denen der DFT anzugleichen. Diese Verfahren sind in [19] einzusehen.
Die Vorraussetzung für diese Verfahren ist jedoch, dass alle Spektrallinien berechnet werden, damit diese zur Korrektur genutzt werden können.

Derartige Transformationen können demnach nicht in dieser Arbeit nicht genutzt werden, da nur einige wenige Spektrallinien berechnet werden sollen.

3.5. Zielsetzung der Arbeit

In dieser Arbeit wird untersucht, inwiefern eine alternative Transformation auf Basis von Rechtecksignalen für die spezifische Anwendung an ABS-Sensorsignalen entworfen werden kann. Hierfür soll jedoch nur eine reduzierte Zahl an Spektrallinien berechnet werden, die die Möglichkeit zur spektralen Korrektur einschränkt. Es soll ein Spektrum berechnet werden, das mit dem der reduzierten DFT vergleichbar ist, um abschließend aus diesem die Verzerrungsmaße bestimmen zu können.

Für dieses Ziel soll ein Verfahren entworfen und anhand von typischen synthetisierten und realen Messsignalen entsprechend getestet werden, damit die Anwendbarkeit an ABS-Sensorsignalen verifiziert werden kann. Zudem sollen die Limitierungen und möglicherweise enstehende Probleme herausgearbeitet werden.

Eine exemplarische Implementierung in VHDL soll weiterhin dazu beitragen, durch Syntheseergebnisse zu zeigen, dass durch die entstandene Implementierung tatsächlich Hardwareressourcen eingespart werden können. Zweckdienlich wird der Cadence Encounter RTL Compiler genutzt. Er gibt Aufschluss darüber, welche Logikressourcen in einer Chip-Implementierung im 350nm Prozess von *austriamicrosystems* genutzt werden würden (siehe Masterarbeit Sabotta [27]).

Abschließend soll eine wertende Analyse der Ergebnisse, die kritisch die Vor- und Nachteile der alternativen Transformation abwägt sowie ein kurzer Ausblick mit Anregungen für weitere Arbeiten gegeben werden.

4. Konzept und Entwurf des Verfahrens

Im Rahmen dieses Kapitels wird das Verfahren zur Spektralschätzung auf Basis von Rechteckfunktionen entworfen. Die grundsätzliche Idee, die zu dieser Transformation führte, soll nachfolgend genauer erläutert werden. Anschließend werden die mathematischen Grundlagen dieses neuen Verfahrens zur Spektralschätzung ausführlich dargestellt.

Im letzten Abschnitt folgen genauere Betrachtungen der Eigenschaften und Limitierungen dieses Verfahrens.

4.1. Rectangular Approximated Fourier-Transform (RAFT)

Die Berechnung der reduzierten DFT ist in einem System aus digitaler Hardware sehr aufwändig. Viele komplexe Operationen müssen durchgeführt werden, um die einzelnen Spektrallinien zu erhalten. Deswegen bestand die Idee diese Berechnung zu optimieren in Hinblick auf die Einsparung von Hardwareressourcen und der damit verbundenen Fläche auf dem Chip (siehe Abbildung 4.1).

Der dazu gewählte Ansatz für die Reduktion von Hardwareressourcen basiert darauf, die Lookup-Tabellen (LUT) für die Cosinus- und Sinusfunktionen in Bezug auf ihre Bitbreite stark zu verringern. Daraus folgt, dass ein Hardwaremultiplizierer mit geringeren Bitbreiten eingesetzt werden kann. Viele kombinatorische Logikelemente können dadurch eingespart werden. In der aktuell verwendeten Realisierung (siehe Dreschoff [3]) wird eine 10-Bit Lookup-Tabelle für die Werte der trigonometrischen Funktionen eingesetzt. Mittels einer geschickten Realisierung durch intelligente Adressierung dieser Tabelle muss nur eine Viertelperiode einer Funktion (Sinus oder Cosinus) in der LUT abgelegt werden. Hieraus werden über Offsetaddition der Adressen und Inversion der Tabelleneinträge alle Funktionswerte beider trigonometrischen Funktionen (Sinus und Cosinus) von Null bis 2π erzeugt. Bei 64 Abtastwerten bedeutet dies, dass genau 16 Werte mit jeweils 10-Bit Auflösung in einer Lookup-Tabelle abzulegen sind. Weiterhin benötigt man zu dieser LUT noch die notwendige Adressierungslogik in Hinblick auf die konkrete Implementierung.

Schafft man es die Lookup-Tabelle auf 1-Bit zu reduzieren ensteht ein sehr großes Einsparungspotential. Der eingesetzte Hardwaremultiplizierer kann durch diese Reduktion gänzlich eingespart werden. Gleichzeitig reduziert sich dadurch der Bedarf an Logikressourcen für die LUT auf ein Minimum. Diese Überlegung führt dazu, dass lediglich Additions- und Subtraktionsoperationen zum Erhalt einer Spektralschätzung durchgeführt werden müssen.

Es existieren bereits einige ähnliche Ansätze für diese *grob quantisierte-DFT* Berechnung. Sie setzen jedoch voraus, dass alle Spektrallinien der DFT (N-Stück) berechnet werden müssen (siehe [19]: *Arithmetische Fourier-Transformation* (AFT) und *grob quantisierte DFT* (QDFT)). Mit dieser Arbeit wird hingegen eine Approximation auf Basis der reduzierten DFT entworfen. Dementsprechend wird mit nur wenigen Spektrallinien gearbeitet, die dann für die anschließenden spektralen Korrekturen eingesetzt werden. Das angestrebte Verfahren nutzt die Fourier-Reihenentwicklung der Transformationsmatrizen, um diese Korrekturen durchzuführen und abschließend ein zur DFT ähnliches Spektrum zu erhalten.

Das allgemeine Vorgehen einer Approximation des DFT-Spektrums unter Verwendung von alternativen Transformationen kann in Matrixschreibweise durch zwei einzelne Transformationen generalisiert dargestellt werden. Der DFT-Vektor $\mathbf{X}$ wird durch die Matrixmultiplikation eines Transformationskerns $\mathbf{W}$ mit dem Eingangsvektor x berechnet (Gleichung 4.1).

$$\mathbf{X} = \mathbf{W} \cdot \mathbf{x} \tag{4.1}$$

Gesucht werden nun zwei Matrizen, die es ermöglichen $\mathbf{X}$ mittels einer Transformationsmatrix $\mathbf{T}$ und einer Korrekturmatrix $\mathbf{K}$ zu ermitteln. Das Ziel dabei ist es eine Transformationsmatrix $\mathbf{T}$ zu finden, welche sehr elegant nur mit Additionen und Subtraktionen berechnet werden kann. Zusätzlich ist es anstrebenswert eine sehr dünnbesetzte Korrekturmatrix $\mathbf{K}$ zu finden, die anschließend eine gute Approximation der DFT liefert (siehe Gleichungen 4.2 und 4.3). Dem Leser wird empfohlen sich mit Kapitel 11 aus [19] vertraut zu machen, um einen guten Einstieg in die Thematik zu finden.

$$\mathbf{X} = \mathbf{W} \cdot \mathbf{x} \tag{4.2}$$
$$\approx \mathbf{K} \cdot (\mathbf{T} \cdot \mathbf{x}) \tag{4.3}$$

Zuerst sollen nun ein paar Begrifflichkeiten definiert werden, die zum besseren Verständnis beitragen. Sie werden einheitlich in dieser Masterarbeit verwendet und sind weiterhin im Glossar einzusehen.

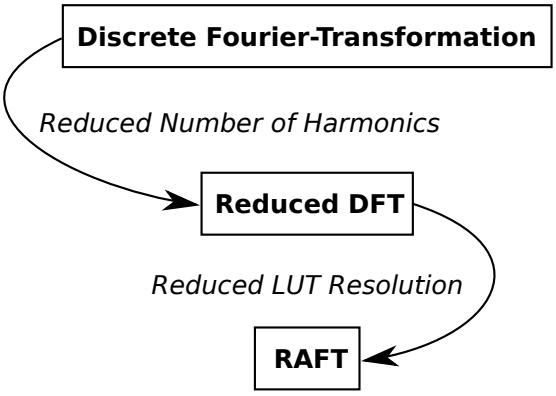


Abbildung 4.1.: Entwicklung der RAFT Idee, durch sukzessive Aufwandsreduktion. Angefangen bei der DFT wird die Anzahl der Harmonischen auf ein Minimum reduziert. Dies führte zur reduzierten DFT. Den nächsten Schritt bildet die Reduktion der Lookup-Tabelle auf 1-Bit Auflösung und damit verbunden die Enstehung der RAFT-Idee.

Die Begrifflichkeiten

$$
\begin{aligned}
N &:= \text{Anzahl der Samples} \\
K &:= \text{Anzahl der Harmonischen} \\
\\
n &:= \text{Zeit- bzw. Winkelindex} \\
k &:= \text{Frequenz- bzw. Ordnungsindex} \\
\\
x_n &:= \text{Relle diskrete Eingangsfolge } 0 \leq n < N \\
X &:= \text{DFT-Transformierte von } x_n \\
R &:= \text{RAFT-Transformierte von } x_n \\
\\
\textbf{fette gedruckte Schrift} &:= \text{Matrix bzw. Vektordarstellung} \\
\\
\mathbf{A} &:= \text{Verzerrungsmatrix Realteil} \\
\mathbf{B} &:= \text{Verzerrungsmatrix Imaginärlteil} \\
\mathbf{A}^{-1} &:= \text{Korrekturmatrix Realteil} \\
\mathbf{B}^{-1} &:= \text{Korrekturmatrix Imaginärteil}
\end{aligned}
\tag{4.4}
$$

4.1.1. Definition der Rechteckfunktionen

Die Substitution der Cosinus- und Sinusfunktionen durch geeignete Rechteckfunktionen sind der Ausgangspunkt für das neue Verfahren. Aus diesem Grund definieren wir analog zu den Walsh-Kaczmarz-Funktionen (Walsh-Sinus: *sir* und Walsh-Cosinus: *cor* siehe [4]) nun äquivalente binärwertige Funktionen: den binären-Sinus (*sib*) und dementsprechend dazu auch den binären-Cosinus (*cob*). Von der Binärwertigkeit ist hierbei der Sonderfall $\mathrm{sib}(x,0)$ ausgenommen, weil dieser nachträglich eingeführt wurde, um die Berechnung des Gleichanteils zu vereinfachen. Diesen Sonderfall ausgenommen können die Funktionen nur die Werte $+1$ und -1 annehmen (siehe Abbildung 4.2).

Mit Hilfe dieser beiden Funktionen wird dann im Folgenden die DFT Berechnung approximiert werden.

$$\mathrm{cob}(x,n) = (-1)^{\lfloor 2\cdot n\cdot x+\frac{1}{2}\rfloor} \tag{4.5}$$

$$\mathrm{sib}(x,n) = \begin{cases} 0 & n=0 \\ (-1)^{\lfloor 2\cdot n\cdot x\rfloor} & n>0 \end{cases} \tag{4.6}$$

Diese Funktionen haben für $n>0$ genau n-Perioden auf dem halboffenen Intervall $[0,1) = \{x\in\mathbb{R}\mid a\leq x<b\}$. Sie sind sehr gut mit den trigonometrischen Basisfunktionen $\sin(2\pi n\cdot x)$ bzw. $\cos(2\pi n\cdot x)$ im selben Intervall (siehe Abbildung 4.2) zu vergleichen.

Der entscheidende Unterschied zwischen dem Walsh-Sinus: *sir* und der neuen Funktion *sib*, liegt wie bereits erwähnt in der Definition der Funktion für $n=0$. Diese Definition ist jedoch notwendig, um die Gleichanteilskomponente des Signals direkt zu berechnen, ohne Korrekturen vornehmen zu müssen.

Abgrenzung zu den Rademacherfunktionen

Die n-ten Rademacherfunktionen werden in [4] definiert als:

$$\mathrm{sir}(2^n\cdot x) \tag{4.7}$$

$$\mathrm{cor}(2^n\cdot x) \tag{4.8}$$

Mit dieser Definition ist nur möglich eine Anzahl von Perioden auf dem Intervall $[0,1)$ anzulegen, die eine Potenz von zwei ist. Für diese Funktionen kann sehr einfach gezeigt werden, dass diese ein Orthogonalsystem bilden (siehe [4]). Das wäre für unsere Transformation ein erstrebenswertes Ziel. Durch die Einschränkungen bezüglich der Anzahl der Perioden auf diesem Intervall ensteht jedoch ein großer Nachteil. Es stellte sich heraus, dass es so lediglich möglich ist nur die 2^n. Harmonischen zu berechnen.

Aus diesem Grund können die reinen Rademacherfunktionen nicht genutzt werden, sodass die oben genannte Neudefinition der Rechteckfunktionen erforderlich wurde.

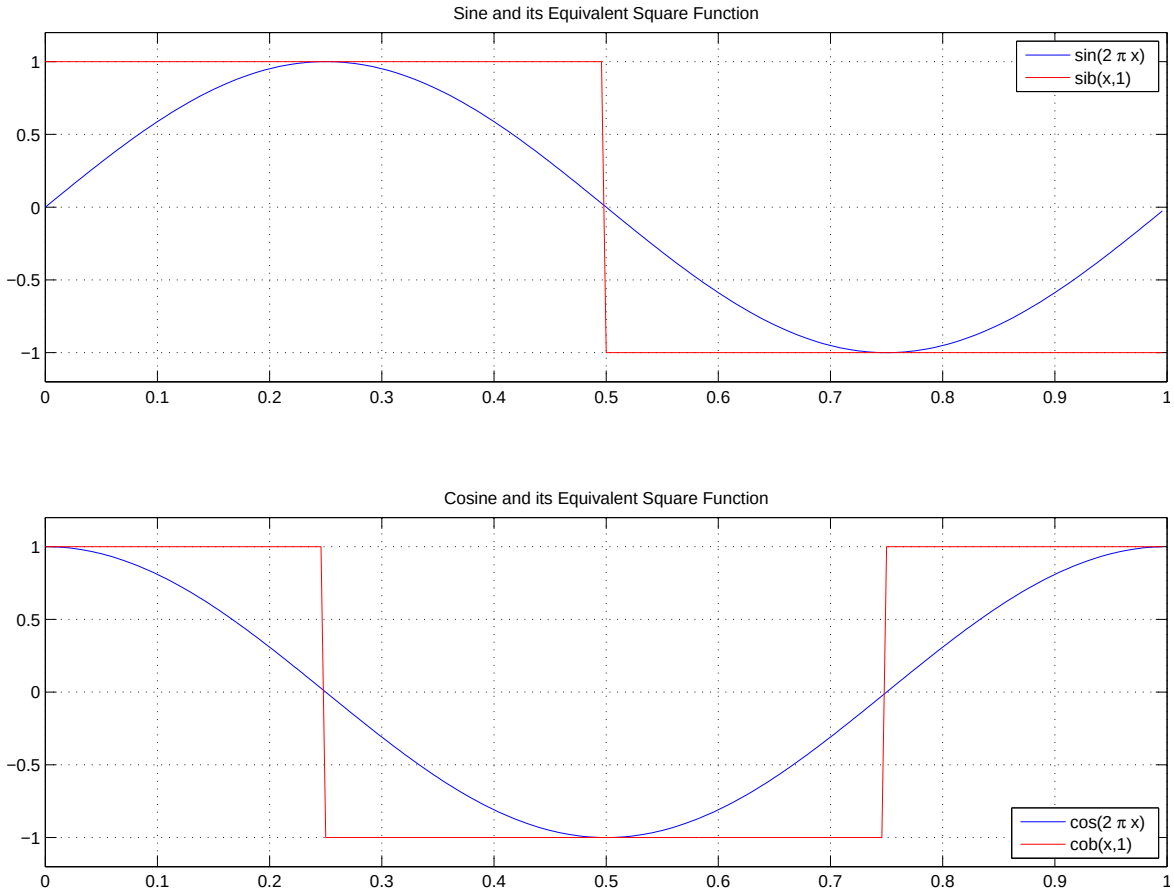


Abbildung 4.2.: Vergleich der Rechteckfunktionen mit den trigonometrischen Ursprungsfunktionen, mit Einpassung der beiden Funktionstypen auf das Intervall $[0, 1)$.

4.1.2. Berechnung des RAFT-Ergebnisvektors

Der RAFT-Ergebnisvektor ensteht durch die Multiplikation des Eingangssignales mit den Rechteckfunktionen. Dieser Vektor wird äquivalent zur DFT definiert und in den folgenden Abschnitten korrigiert, um das DFT Spektrum zu approximieren.

Berechnung der DFT eines reellen Eingangssignals

Die Idee hinter der RAFT verfolgt das Ziel die Lookup-Tabelle auf 1-Bit zu reduzieren. Dies entspricht der Reduktion der trigonometrischen Funktionen auf Rechteckfunktionen. Wir beginnen mit der Definition der DFT zur Nachvollziehung der RAFT-Idee.

Für reelle Eingangsfolgen x_n mit N-Samples lässt sich die DFT wie folgt in jeweils den Real- und Imaginärteil zerlegen:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-j2\pi \frac{k}{N} \cdot n} \tag{4.9}$$

$$X_k = \left(\sum_{n=0}^{N-1} x_n \cdot \cos\left(2\pi \frac{k}{N} \cdot n\right) - j \cdot \sum_{n=0}^{N-1} x_n \cdot \sin\left(2\pi \frac{k}{N} \cdot n\right) \right) \tag{4.10}$$

$$X_k = \mathrm{Re}\{X_k\} + j \cdot \mathrm{Im}\{X_k\} \tag{4.11}$$

Matrixdarstellung der reduzierten DFT

Die Matrixschreibweise eigenet sich besonders gut dazu die DFT darzustellen. Dementsprechend muss nun eine Matrix mit den entsprechenden Drehfaktoren aufgestellt werden (siehe [32]).

Definieren wir jetzt die Drehfaktorfunktion $w(n)$ und mit dieser die Drehfaktormatrix $\mathbf{W}$ sowie den Spaltenvektor mit den reellen Abtastwerten des Eingangssignals $\mathbf{x}$. Dann erhalten wir eine sehr einfache Matrixdarstellung der DFT:

$$w(n) \;=\; e^{-j\frac{2\pi n}{N}} \;=\; cos\left(\frac{2\pi n}{N}\right) - j \cdot sin\left(\frac{2\pi n}{N}\right) \tag{4.12}$$

$$\mathbf{W} = \begin{bmatrix} w(0) & w(0) & w(0) & \dots & w(0) \\ w(0) & w(1\cdot 1) & w(1\cdot 2) & \dots & w(1\cdot(N-1)) \\ w(0) & w(1\cdot 2) & w(2\cdot 2) & \dots & w(2\cdot(N-1)) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w(0) & w(1\cdot(N-1)) & w(2\cdot(N-1)) & \dots & w(K\cdot(N-1)) \end{bmatrix} \tag{4.13}$$

$$\mathbf{x} = \begin{bmatrix} x_0 & x_1 & x_2 & \dots & x_{N-1} \end{bmatrix}^T \tag{4.14}$$

Mit diesen Definitionen erfolgt unmittelbar die Matrixdarstellung der DFT zu:

$\mathbf{X} = \mathbf{W} \cdot \mathbf{x}$

In expandierter Form erhält man dafür den folgenden Ausdruck:

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_K \end{bmatrix} = \begin{bmatrix} w(0) & w(0) & w(0) & \dots & w(0) \\ w(0) & w(1\cdot 1) & w(1\cdot 2) & \dots & w(1\cdot(N-1)) \\ w(0) & w(1\cdot 2) & w(2\cdot 2) & \dots & w(2\cdot(N-1)) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w(0) & w(1\cdot(N-1)) & w(2\cdot(N-1)) & \dots & w(K\cdot(N-1)) \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix} \tag{4.15}$$

Zur Zerlegung dieses Ausdrucks in eine nach Real- und Imaginärteil getrennte Form kann man die Drehfaktormatrix $\mathbf{W}$ in den Real- und Imaginärteil aufspalten. Durch diese Zerlegung erhalten wir eine Cosinus- und eine Sinusmatrix, welche jeweils mit dem Spaltenvektor $\mathbf{x}$ multipliziert werden:

$$\mathbf{X} = \mathbf{W} \cdot \mathbf{x} \tag{4.16}$$
$$= \mathrm{Re}\{\mathbf{W}\} \cdot \mathbf{x} + j \cdot \mathrm{Im}\{\mathbf{W}\} \cdot \mathbf{x} \tag{4.17}$$
$$= \mathbf{Cos} \cdot \mathbf{x} \;-\; j \cdot \mathbf{Sin} \cdot \mathbf{x} \tag{4.18}$$

Die Definition des RAFT-Ergebnisvektors

Definieren wir nun den RAFT-Ergebnisvektor mit den eigens definierten *cob*- bzw. *sib*-Funktionen, so erhalten wir die RAFT-Transformierte R_k. Dies geschieht analog zur DFT unter Verwendung der binären Funktionen.

$$R_k = \left(\sum_{n=0}^{N-1} x_n \cdot \mathrm{cob}\left(\frac{n}{N}, k\right) - j \cdot \sum_{n=0}^{N-1} x_n \cdot \mathrm{sib}\left(\frac{n}{N}, k\right) \right) \tag{4.19}$$

$$R_k = \mathrm{Re}\{R_k\} + j \cdot \mathrm{Im}\{R_k\} \tag{4.20}$$

Zerlegt man nun die RAFT-Transformierte R_K in den Real- und Imaginärteil, so erhält man:

$$\mathrm{Re}\{R_k\} = \sum_{n=0}^{N-1} x_n \cdot \mathrm{cob}\left(\frac{n}{N}, k\right) \tag{4.21}$$

$$\mathrm{Im}\{R_k\} = -\sum_{n=0}^{N-1} x_n \cdot \mathrm{sib}\left(\frac{n}{N}, k\right) \tag{4.22}$$

Matrixdarstellung der RAFT

Die RAFT kann ebenfalls in Matrixschreibweise dargestellt werden, wie es auch bei der DFT möglich ist. Ausgehend von der DFT (in Matrixschreibweise) substituieren wir die **Cos**- sowie die **Sin**-Matrix durch deren binäre Pendants (siehe Formeln 4.18 und 4.13 f.).

$$v(n,k) \;=\; cob\left(\frac{n}{N}, k\right) - j \cdot sib\left(\frac{n}{N}, k\right) \tag{4.23}$$

$$\mathbf{V} = \begin{bmatrix} v(0,0) & v(1,0) & v(2,0) & \dots & v((N-1),0) \\ v(0,1) & v(1,1) & v(2,1) & \dots & v((N-1),1) \\ v(0,2) & v(1,2) & v(2,2) & \dots & v((N-1),2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v(0,K) & v(1,K) & v(2,K) & \dots & v((N-1),K) \end{bmatrix} \tag{4.24}$$

$$\mathbf{R} = \mathbf{V} \cdot \mathbf{x} \tag{4.25}$$
$$= \mathrm{Re}\{\mathbf{V}\} \cdot \mathbf{x} + j \cdot \mathrm{Im}\{\mathbf{V}\} \cdot \mathbf{x} \tag{4.26}$$
$$= \mathbf{Cob} \cdot \mathbf{x} - j \cdot \mathbf{Sib} \cdot \mathbf{x} \tag{4.27}$$

Fourier-Reihenentwicklung der RAFT

Die Fourier-Reihenentwicklung von Rechteckfunktionen ist einfach zu bestimmen und ist auch in vielen Formelsammlungen zu finden (siehe z.B. [20]). Wendet man diese nun auf die *cob*- bzw. *sib*-Funktionen an, so können sie wie folgt dargestellt werden (für alle $k > 0$).

$$\mathrm{cob}\left(\frac{n}{N}, k\right) \approx \frac{4}{\pi} \sum_{i=0}^{N-1} (-1)^i \cdot \frac{\cos(2\pi\ (2i+1)\ k \cdot \frac{n}{N})}{2i+1} \tag{4.28}$$
$$\approx \frac{4}{\pi} \left(\cos\left(2\pi\ k \cdot \frac{n}{N}\right) - \frac{\cos(2\pi\ 3k \cdot \frac{n}{N})}{3} + \frac{\cos(2\pi\ 5k \cdot \frac{n}{N})}{5} - \cdots\right) \tag{4.29}$$

$$\mathrm{sib}\left(\frac{n}{N}, k\right) \approx \frac{4}{\pi} \sum_{i=0}^{N-1} \frac{\sin(2\pi\ (2i+1)\ k \cdot \frac{n}{N})}{2i+1} \tag{4.30}$$
$$\approx \frac{4}{\pi} \left(\sin\left(2\pi\ k \cdot \frac{n}{N}\right) + \frac{\sin(2\pi\ 3k \cdot \frac{n}{N})}{3} + \frac{\sin(2\pi\ 5k \cdot \frac{n}{N})}{5} + \cdots\right) \tag{4.31}$$

Die *cob*-Funktion ist eine zur Ordinate spiegelsymetrische Rechteckfunktion mit k-Perioden auf dem Intervall $[0,1)$. Die *sib*-Funktion hingegen ist eine zur Ordinate punktsymetrische Rechteckfunktion, ebenfalls mit k-Perioden auf dem Intervall $[0,1)$. Deswegen enstehen zwei sehr unterschiedliche Fourier-Reihenentwicklungen für diese beiden Funktionen.

Nehmen wir nun diese beiden Fourier-Reihenentwicklungen und setzen diese in den Real- bzw. Imaginärteil von R_k ein, so erhalten wir eine Darstellung in der wir die tatsächlich in R_k enthaltenen harmonischen Anteile erkennen.

$$\mathrm{Re}\{R_k\} = \sum_{n=0}^{N-1} x_n \cdot \mathrm{cob}\left(\frac{n}{N}, k\right) \tag{4.32}$$

$$\approx \sum_{n=0}^{N-1} x_n \cdot \frac{4}{\pi} \sum_{i=0}^{N-1} (-1)^i \cdot \frac{\cos(2\pi\ (2i+1)\ k \cdot \frac{n}{N})}{2i+1} \tag{4.33}$$

$$\approx \frac{4}{\pi} \sum_{n=0}^{N-1} x_n \cdot \left(\underbrace{\cos\left(2\pi\ k \cdot \frac{n}{N}\right)}_{\text{gewünschter Term}} - \underbrace{\frac{\cos(2\pi\ 3k \cdot \frac{n}{N})}{3}}_{\text{1. Fehlerterm}} + \underbrace{\frac{\cos(2\pi\ 5k \cdot \frac{n}{N})}{5}}_{\text{2. Fehlerterm}} - \cdots \right) \tag{4.34}$$

Diese Formeln zeigen, dass ein relativ großer Fehler in Bezug auf die DFT für jede Komponente des Vektors R_k durch die RAFT-Transformation entsteht. Jede Spektralkomponente enthält weitere harmonische Anteile, die dieser unterlagert sind. Mit dem Ziel eine elegante Hardwareimplementierung mit geringem Stromverbrauch und Ressourcenverbrauch zu erhalten, nehmen wir diesen Fehler jedoch hin und versuchen ihn in einem weiteren Schritt durch die bereits berechneten Zwischenergebnisse zu minimieren.

Entwickelt man der vollständigkeit halber dementsprechend analog dazu auch den Imaginärteil der Transformation, so erhalten wir für diesen ebenfalls die Fourier-Reihenentwicklung:

$$\mathrm{Im}\{R_k\} = -\sum_{n=0}^{N-1} x_n \cdot \mathrm{sib}\left(\frac{n}{N}, k\right) \tag{4.35}$$

$$\approx -\sum_{n=0}^{N-1} x_n \cdot \frac{4}{\pi} \sum_{i=0}^{N-1} \frac{\sin(2\pi\ (2i+1)\ k \cdot \frac{n}{N})}{2i+1} \tag{4.36}$$

$$\approx -\frac{4}{\pi} \sum_{n=0}^{N-1} x_n \cdot \left(\underbrace{\sin\left(2\pi\ k \cdot \frac{n}{N}\right)}_{\text{gewünschter Term}} + \underbrace{\frac{\sin(2\pi\ 3k \cdot \frac{n}{N})}{3}}_{\text{1. Fehlerterm}} + \underbrace{\frac{\sin(2\pi\ 5k \cdot \frac{n}{N})}{5}}_{\text{2. Fehlerterm}} + \cdots \right) \tag{4.37}$$

Folglich nehmen wir an, dass die **Cob**- und **Sib**-Matrizen implizit zusätzliche Anteile von Harmonischen enthalten, welche kompensiert werden müssen, um ein zur reduzierten DFT ähnliches Spektrum zu erhalten. Im folgenden Abschnitt wird gezeigt werden, wie diese Fehler nun unter Verwendung von Zwischenergebnissen kompensiert werden können.

4.1.3. Approximation der reduzierten DFT durch Korrektur des RAFT-Ergebnisvektors

Durch die RAFT-Transformation wird ein Ergebnisvektor R_k berechnet. Dieser ist jedoch noch nicht direkt mit dem Ergebnisvektor der DFT (X_k) vergleichbar. Zur Kompensation versuchen wir die zusätzlichen Cosinus- bzw. Sinusschwingungen die im Ergebnisvektor auftreten durch bereits vorhandene Zwischenergebnisse mit einer Rücksubstitution wie folgt zu eliminieren.

Exemplarisch wird auch hier die Berechnung der Ergebnisse anhand des Realteils gezeigt, da sich die Berechnung des Imaginärteils aus diesem ableiten lässt.

Die RAFT-Transformation lieferte uns folgende Matrixdarstellung des Realteils unseres Ergebnisvektors R_k:

$$\mathrm{Re}\left\{\begin{bmatrix} R_0 \\ R_1 \\ \vdots \\ R_k \end{bmatrix}\right\} = \begin{bmatrix} \mathrm{cob}\left(\frac{0}{N},0\right) & \mathrm{cob}\left(\frac{1}{N},0\right) & \cdots & \mathrm{cob}\left(\frac{N-1}{N},0\right) \\ \mathrm{cob}\left(\frac{0}{N},1\right) & \mathrm{cob}\left(\frac{1}{N},1\right) & \cdots & \mathrm{cob}\left(\frac{N-1}{N},1\right) \\ \vdots & \cdots & \ddots & \vdots \\ \mathrm{cob}\left(\frac{0}{N},k\right) & \mathrm{cob}\left(\frac{1}{N},k\right) & \cdots & \mathrm{cob}\left(\frac{N-1}{N},k\right) \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} \tag{4.38}$$

$$= \begin{bmatrix} \sum_{n=0}^{N-1} x_n \cdot \mathrm{cob}\left(\frac{n}{N},0\right) \\ \vdots \\ \sum_{n=0}^{N-1} x_n \cdot \mathrm{cob}\left(\frac{n}{N},k\right) \end{bmatrix} \tag{4.39}$$

$$= \mathbf{Cob} \cdot \mathbf{x} \tag{4.40}$$

$$\tag{4.41}$$

Am Ende der Transformation existieren also sämtliche Multiplikationen des Eingangsvektors x_k mit $\mathrm{cob}(\frac{n}{N},k)$im Bereich $0 < k \leq K$. Die Idee der Korrektur besteht nun darin diese Zwischenergebnisse zu nutzen, um den Ausgangsvektor zu korrigieren.

Definieren wir nun $\hat{R}_k$ als die Fourier-Reihenentwicklung aus dem vorigen Abschnitt (siehe Gleichungen 4.33 und 4.36) wie folgt, so können wir weitere Untersuchungen durchführen.

$$\mathrm{Re}\{\hat{R}_k\} = \sum_{n=0}^{N-1} x_n \cdot \frac{4}{\pi} \sum_{i=0}^{N-1} (-1)^i \cdot \frac{\cos(2\pi\ (2i+1)\ k \cdot \frac{n}{N})}{2i+1} \tag{4.42}$$

$$= \frac{4}{\pi} \sum_{n=0}^{N-1} x_n \cdot \left(\cos\left(2\pi\ k \cdot \frac{n}{N}\right) - \frac{\cos(2\pi\ 3k \cdot \frac{n}{N})}{3} + \frac{\cos(2\pi\ 5k \cdot \frac{n}{N})}{5} - \cdots\right) \tag{4.43}$$

$$\mathrm{Im}\{\hat{R}_k\} = -\sum_{n=0}^{N-1} x_n \cdot \frac{4}{\pi} \sum_{i=0}^{N-1} \frac{\sin(2\pi\ (2i+1)\ k \cdot \frac{n}{N})}{2i+1} \tag{4.44}$$

$$= -\frac{4}{\pi} \sum_{n=0}^{N-1} x_n \cdot \left(\sin\left(2\pi\ k \cdot \frac{n}{N}\right) + \frac{\sin(2\pi\ 3k \cdot \frac{n}{N})}{3} + \frac{\sin(2\pi\ 5k \cdot \frac{n}{N})}{5} + \cdots \right) \tag{4.45}$$

Der Vektor $\hat{R}_k$ kann nun wie folgt in Matrixschreibweise dargestellt werden. Wir nehmen dafür an, dass zwei Matrizen (**A** und **B**) aufgestellt werden können, die die zusätzlichen harmonischen Anteile im Ergebnisvektor beschreiben.

$$\mathrm{Re}\left\{\hat{\mathbf{R}}\right\} = \quad \mathbf{A} \cdot (\ \mathbf{Cos} \cdot \mathbf{x}\ ) \tag{4.46}$$

$$\mathrm{Im}\left\{\hat{\mathbf{R}}\right\} = -\mathbf{B} \cdot (\ \mathbf{Sin} \cdot \mathbf{x}\ ) \tag{4.47}$$

Sind die Matrizen **A** und **B** invertierbar, so können deren inverse Matrizen theoretisch also linksseitig an die Gleichungungen heranmultipliziert werden, um diese zu kompensieren.
Da wir jedoch durch die RAFT nicht exakt diese Gleichungen erhalten haben, sondern nur den Vektor **R**, können sie nicht vollständig kompensiert werden. Der Ansatz, den wir wählen, besteht also darin, diese Korrekturmatrizen linksseitig an den Real- bzw. Imaginärteil unseres Ergebnisvektors **R** zu multiplizieren.
Dadurch erhalten wir jedoch implizit eine Kompensation des Ergebnisses mit weiteren *cob*- bzw. *sib*-Funktionen und nicht mit Cosinus- und Sinusschwingungen wie es bei $\hat{R}$ der Fall ist. Demnach kann eine perfekte Kompensation niemals erreicht werden.

Auf den ersten Blick erscheint dies nun wesentlich aufwändiger zu sein. In der Realität ist dies in Hinblick auf die Hardwareimplementierung jedoch nicht der Fall. Die inneren Matrixmultiplikationen können bedingt durch die Eigenschaften der **Cob**- bzw. **Sib**-Matrix komplett durch Additionen und Subtraktionen gelöst werden. Die Multiplikation mit den Korrekturmatrizen $\mathbf{A}^{-1}$ und $\mathbf{B}^{-1}$ sind ebenfalls zu vernachlässigen, da diese Matrizen sehr dünn besetzt sind.

Beispielhaft sind hier die Matrizen für die Korrektur von 7 Harmonischen angeführt.

$$\mathbf{A} = \frac{4}{\pi}\begin{bmatrix} \frac{\pi}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\frac{1}{3} & 0 & \frac{1}{5} & 0 & -\frac{1}{7} \\ 0 & 0 & 1 & 0 & 0 & 0 & -\frac{1}{3} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{B} = \frac{4}{\pi}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{1}{3} & 0 & \frac{1}{5} & 0 & \frac{1}{7} \\ 0 & 0 & 1 & 0 & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.48}$$

$$\mathbf{A}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{\pi}{4} & 0 & \frac{\pi}{12} & 0 & -\frac{\pi}{20} & 0 & \frac{\pi}{28} \\ 0 & 0 & \frac{\pi}{4} & 0 & 0 & 0 & \frac{\pi}{12} & 0 \\ 0 & 0 & 0 & \frac{\pi}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\pi}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{\pi}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{\pi}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\pi}{4} \end{bmatrix} \quad \mathbf{B}^{-1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{\pi}{4} & 0 & -\frac{\pi}{12} & 0 & -\frac{\pi}{20} & 0 & -\frac{\pi}{28} \\ 0 & 0 & \frac{\pi}{4} & 0 & 0 & 0 & -\frac{\pi}{12} & 0 \\ 0 & 0 & 0 & \frac{\pi}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\pi}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{\pi}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{\pi}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\pi}{4} \end{bmatrix} \tag{4.49}$$

Außerdem können in dieser Berechnung die Matrizen weiter vereinfacht werden, da für die Ermittlung des HD_5-Verzerrungsmaßes die Berechnung des Gleichanteils nicht notwendig ist. Somit können die Korrekturmatrizen von der Größe $(K+1) \times (K+1)$ auf $K \times K$ verkleinert werden.

$$\begin{aligned} \mathrm{Re}\{\mathbf{X}\} &= \mathbf{Cos} \cdot \mathbf{x} \\ &\approx \mathbf{A}^{-1} \cdot \mathrm{Re}\{\mathbf{R}\} \end{aligned} \tag{4.50}$$

$$\begin{aligned} \mathrm{Im}\{\mathbf{X}\} &= -\,\mathbf{Sin} \cdot \mathbf{x} \\ &\approx \mathbf{B}^{-1} \cdot \mathrm{Im}\{\mathbf{R}\} \end{aligned} \tag{4.51}$$

$$\mathbf{X} \approx \mathbf{A}^{-1} \cdot \mathrm{Re}\{\mathbf{R}\} + j \cdot \mathbf{B}^{-1} \cdot \mathrm{Im}\{\mathbf{R}\} \tag{4.52}$$

Mit diesen letzten Gleichungen wird die Approximation des Ergebnisvektors $\mathbf{X}$ erreicht (siehe 4.52). Folglich erhalten wir ein zur reduzierten DFT ähnliches Spektrum, das über ein alternatives Verfahren berechnet wurde.

4.2. Fehlerbetrachtung

Es verbleibt durch diese Approximation ein Restfehler, obwohl eine Korrektur des RAFT-Ergebnisvektors durchgeführt wurde. Für jede Zeile der RAFT-Matrix wird eine Reihenapproximation mit einer unbegrenzten Anzahl an Glieder genutzt. Es stehen jedoch nur eine sehr begrenzte Anzahl an Korrekturtermen zur Verfügung. Dies hat zur Folge, dass immer ein Restfehler verbleibt, der jedoch auf ein Minimum reduziert werden kann, sofern das Eingangssignal sehr stark bandbegrenzt wird.

4.2.1. Amplitudenfehler

Der Amplitudenfehler ist direkt vom Eingangssignal abhängig, wie man durch die Fortsetzung der Fehlerterme in der Reihenentwicklung sehen konnte. Untersucht man jedoch ein sehr stark bandbegrenztes Signal mit der RAFT, so kann man zeigen, dass der entstehende Fehler nur minimal ist. Hat ein Testsignal nur Spektrallinien im zu untersuchenden Bereich (es gilt also: $H_k = 0$ für $k > K$; K ist die Anzahl der zu Analysierenden), so kann angenommen werden, dass für die Multiplikationen der Eingangsfolge x_n mit den *cob*- und *sib*-Funktionen gilt:

$$x_n \cdot cob\left(\frac{n}{N}, k\right) = 0 \quad \forall \quad k > K \tag{4.53}$$

$$x_n \cdot sib\left(\frac{n}{N}, k\right) = 0 \quad \forall \quad k > K \tag{4.54}$$

Dieselbe Aussage lässt sich dementsprechend äquivalent für die DFT formulieren:

$$x_n \cdot cos\left(2\pi \frac{k}{N} \cdot n\right) = 0 \quad \forall \quad k > K \tag{4.55}$$

$$x_n \cdot sin\left(2\pi \frac{k}{N} \cdot n\right) = 0 \quad \forall \quad k > K \tag{4.56}$$

Folglich kann man direkt sehen das folgender Zusammenhang gilt:

$$R_k = 0 \quad \forall \quad k > K \tag{4.57}$$

Durch diesen Zusammenhang kann man nun erkennen, dass für diese sehr stark bandbegrenzten Signale die spektrale Approximation unter Verwendung des RAFT-Verfahrens in

der Theorie sehr gut mit dem tatsächlichen Spektrum übereinstimmt. Die für die Kompensation notwendigen Spektrallinien sind nicht im Testsignal vorhanden und müssen folglich nicht zur Korrektur eingesetzt werden.

Die Einschränkung dabei ist, dass wir in der Theorie ein stark bandbegrenztes Signal besitzen müssen, um die RAFT tatsächlich anwenden zu können. Für die Nutzung des Algorithmus zur Berechnung eines Verzerrungsmaßes an Messdaten wird sich jedoch zeigen, dass der Fehler, der durch ein Signal, welches ein breiteres Spektrum besitzt als es die theoretische Obergrenze erlaubt, keine sehr großen Auswirkungen auf das Verzerrungsmaß hat, solange die Form des Spektrums einigen Randbedingungen folgt.

4.2.2. Phasenfehler

Die mit Rechteckfunktionen approximierten Cosinus- und Sinusfunktionen haben zudem noch eine Phasenabweichung, die sich relativ einfach bestimmen lässt. Diese ist konstant für jede Harmonische und könnte im Postprocessing nötigenfalls entfernt werden.

Die Phasendrehung zwischen zwei Signalen kann in Radiant wie folgt bestimmt werden:

$$\frac{\Delta\varphi}{2\pi} = \frac{\Delta t}{T} = \frac{\Delta\text{Samples}}{\text{Samples pro Periode}} \tag{4.58}$$

Beim direkten Vergleich zwischen den **Cob**- und **Sib**-Matrizen mit den dazugehörigen Cosinus- und Sinusmatrizen fällt auf, dass für jede Funktion eine Verschiebung im Zeitbereich von einem halben Sample auftritt. Jede diskrete *sib*- bzw. *cob*-Funktion eilt, seiner entsprechenden diskreten trigonometrischen Funktion um ein halbes Sample voraus (siehe Abbildung 4.3).

Die Periodendauer der einzelnen Funktionen kann durch $\frac{N}{i}$ ausgedrückt werden, da genau i-Perioden der Musterfunktionen in den Zeilen der Matrix, welche N-Werte breit ist, auftreten. Setzt man dies nun in die Gleichung 4.58 ein, so erhalten wir den folgenden Zusammenhang:

$$\frac{\Delta\varphi}{2\pi} = \frac{\frac{1}{2}}{\frac{N}{i}} \tag{4.59}$$

Damit ergibt sich jeweils eiene Phasenverschiebung von $\Delta\varphi$ für die i-te harmonische Komponente zu:

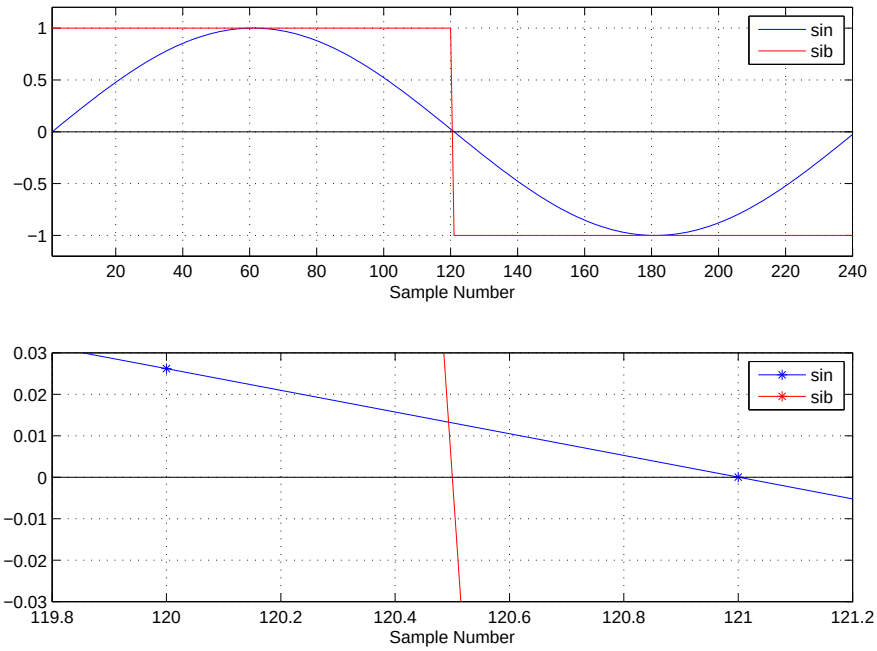


Abbildung 4.3.: Der obere Abschnitt dieser Abbildung zeigt das Verhältnis der diskreten Sinus-Funktion zur *sib*-Funktion. Im unteren Abschnitt (Ausschnitt und Vergrößerung des oberen Teilbildes) ist deutlich sichtbar, dass die Verschiebung der Nullstelle dieser beiden Funktionen genau ein halbes Sample beträgt. Zusätzlich hat die diskrete *sib*-Funktion keine echte Nullstelle, da nur Werte von ± 1 genutzt werden, um diese zu konstruieren.

$$\Delta\varphi = \frac{i \cdot \pi}{N} \tag{4.60}$$

Diese Annahme konnte mittels einer Simulation in MATLAB verifiziert werden, indem 1000 Testsignale mit sechs Harmonischen jeweils mit zufälliger Amplituden- und Phaseinformation erzeugt wurden. Diese Signale wurden anschließend DFT und RAFT transformiert und die Phasendifferenz zwischen den Harmonischen (DFT und RAFT) gebildet. In Übereinstimmung mit der Theorie finden sich Resultate der Simulation in Tabelle 4.1.

In der bisherigen Realisierung der Selbstdiagnosefunktion ist lediglich der Betrag $|H|$ bzw. das Betragsquadrat $|H|^2$ verwendet, sodass die obige Berechnung nur vollständigkeitshalber angeführt ist.

Es gibt bereits Überlegungen die Phaseninformateion für die Analyse des Zustandes auszuwerten. Essentiell ist dafür die Durchführung der genannte Phasenkorrektur nach der Anwendung des RAFT-Verfahrens. Aus diesem Grund wurde diese in Hinblick auf weitere Arbeiten hier erwähnt.

Spektral-komponente	erwartete Phasendifferenz $\Delta\varphi$	simulierte Phasendifferenz $\Delta\varphi$
H_1	$(1 \cdot \pi)/240 \approx 0,01309$	$0,0131$
H_2	$(2 \cdot \pi)/240 \approx 0,02618$	$0,0262$
H_3	$(3 \cdot \pi)/240 \approx 0,03927$	$0,0393$
H_4	$(4 \cdot \pi)/240 \approx 0,05236$	$0,0524$
H_5	$(5 \cdot \pi)/240 \approx 0,06545$	$0,0654$
H_6	$(6 \cdot \pi)/240 \approx 0,07854$	$0,0785$

Tabelle 4.1.: Ein tabellarischer Vergleich zwischen den theoretischen und den mit MATLAB simulierten Phasendifferenzen an Testsignalen.

4.3. Eigenschaften der RAFT

Für weitere Betrachtungen des RAFT-Verfahrens zur Approximation des DFT-Spektrums, ist es notwendig, kurz darauf einzugehen, welches Ziel bei dieser Approximation verfolgt wird. Besonders die nicht triviale Wahl der Abtastrate stellte ein großes Problem dar, welches mittlerweile gelöst werden konnte. Es entstand eine weitere Limitierung, die zur Einschränkung der Anwendbarkeit des RAFT-Verfahrens führte.

4.3.1. Diskrete Orthogonaltransformationen

Durch unseren Ansatz versuchen wir eine diskrete Orthogonaltransformation (die DFT) in eine andere zu überführen.

Dies ist nur begrenzt möglich, da wir durch die Fourier-Reihenentwicklung der *sib*- bzw. *cob*-Funktionen wissen, dass diese eine gewisse Selbstähnlichkeit besitzen und damit nicht zueinander orthogonal sein können. Welche Maßnahmen getroffen werden können, um eine quasi-orthogonale Transformation zu erhalten, wird in diesem Abschnitt näher erläutert.

Orthogonalität der DFT
Mit Hilfe der geometrischen Reihe kann gezeigt werden, dass die DFT eine Orthogonaltransformation ist (siehe [30]).

Die Orthogonalität kann mit der Definition des Standardskalarprodukt im $\mathbb{C}^n$ gezeigt werden:

$$\langle s_k, s_l \rangle := \sum_{n=0}^{N-1} s_k(n)\, \overline{s_l(n)} = 0 \quad \forall \quad l \neq k \tag{4.61}$$

Nutzen wir die geometrische Reihe für den komplexen Transformationskern der DFT (hier $s_k(n)$ bzw. $s_l(n)$):

$$s_{N-1} = \sum_{i=0}^{N-1} q^i \quad \text{mit } q \neq 1 \tag{4.62}$$

$$= \frac{1-q^N}{1-q} \tag{4.63}$$

So kann folgende Zusammenhang aufgestellt werden:

$$\langle s_k, s_l \rangle := \sum_{n=0}^{N-1} e^{-j2\pi kn/N} \, e^{j2\pi ln/N} \tag{4.64}$$

$$= \sum_{n=0}^{N-1} e^{-j2\pi(k-l)n/N} = \frac{1 - e^{-j2\pi(k-l)n}}{1 - e^{-j2\pi(k-l)n/N}} \tag{4.65}$$

Für alle $k \neq l \quad k,l \in \mathbb{N}$ gilt, dass der Zähler zu Null wird, während der Nenner verschieden von Null ist. Für $k = l$ darf man diese Entwicklung der geometrischen Reihe natürlich nicht anwenden. Es gilt nämlich $e^{-j2\pi(k-l)n/N} = 1$ und eine Division durch Null entstünde.

Für diesen speziellen Fall gilt: $\langle s_k, s_l \rangle = N \quad \forall \quad k = l$.

Die Orthogonalität der DFT ist so formal bewiesen. Ein ähnlicher Zusammenhang für das RAFT-Verfahren ist nicht ohne weiteres beweisbar. Es kann zwar gezeigt werden, dass die Rademacherfunktionen der Form $\{\text{sir, cor}\}(2^n \, x)$ ein Orthogonalsystem bilden (siehe [4]), jedoch ist dies im Allgemeinen nicht für alle Rechteckfunktionen der Form $\{\text{sir, cor}\}(x)$ gezeigt worden.

Zur Annäherung an die Orthogonalität können folgende Überlegungen geführt werden, die im Folgenden geschildert werden.

4.3.2. Anpassung der Abtastrate des Eingangssignals

Die empirische Simulation zeigte, eine Minimierung des Fehlers, wenn die Anzahl der Abtastwerte pro Zeitfenster auf eine bestimmte Art und Weise gewählt wird.

Es stellte sich heraus, dass die Anzahl der Abtastwerte ein ganzzahliges Vielfaches der benötigten Perioden der *sib*- bzw. *cob*-Funktionen sein muss. Die eingesetzten Fourier-Reihenentwicklungen können sonst nicht genutzt werden, da die benötigten Symmetrien der Rechteckfunktion sonst nicht gegeben sind. Folglich müssen die Fourier-Reihenentwicklungen möglichst gut mit den verwendeten Rechteckfunktionen übereinstimmen.

Für die Symmetrie der Matrizen definieren wir deshalb zwei Bedingungen:

1. Die **Cob**-Matrix soll orthogonal zur **Sib**-Matrix stehen. Dies kann mit der folgenden Berechnung überprüft werden:
   $\mathbf{Sib} \cdot \mathbf{Cob}^T = \mathbf{Cob} \cdot \mathbf{Sib}^T = \mathbf{0}$
   So gelingt es, dass jede der verwendeten *sib*-Funktionen orthogonal auf jeder der verwendeten *cob*-Funktion steht (analog zur DFT steht also der Imaginärteil orthogonal auf dem Realteil).

2. Die Anzahl der Abtastwerte für die Transformation muss folgende Bedingung erfüllen [1]:
   $N = 4 \cdot lcm(\ 1,\ 2,\ 3,\ \cdots,\ K)$
   Diese sorgt für die Gleichverteilung der Werte ($+1$ und -1) innerhalb der Zeilen der Matrix. Die Matrizen werden somit symmetriert und es wird dafür gesorgt, dass die Fourier-Reihenentwicklung möglichst gut mit den verwendeten Funktionen übereinstimmt. Da vorausgesetzt wird, dass die positive und die negative Halbwelle des Rechtecksignals den Tastgrad von 50 % erfüllen.

Die ersten Tests der RAFT-Idee ergaben, dass bei Nichtbeachtung dieser Bedingungen der enstehende Fehler sehr groß ist und damit das RAFT-Verfahren nicht mehr nutzbar an ABS-Signalen ist (siehe dafür [15]).

Anzahl der Harmonischen K	Anzahl der Samples N
1	4
2	8
3	24
4	96
5	240
6	240
7	7056
8	14784
9	22176
10	22176

Tabelle 4.2.: Zusammenhang zwischen der Anzahl der zu berechnenden Harmonischen und der benötigten Anzahl an Abtastwerten pro Erfassungsfenster.

Die Tabelle 4.2 veranschaulicht, dass die Anwendung der RAFT nur für die Spektralschätzung von ≤ 6 Harmonischen zweckmäßig ist. Die Anzahl der benötigten Abtastwerte pro Zeitfenster wächst sehr stark mit der Anzahl der zu berechnenden Harmonischen. Eine Hardwareimplementierung mit dieser hohen Anzahl an Abtastwerten ergäbe in der Praxis für eine größere Anzahl an Harmonischen ($K > 6$) keine Vorteile im Vergleich zur Implementierung der rDFT.

Eine solche Implementierung führte zu deutlich höheren Systemanforderungen. Die Erhöhung des Systemtakts, sowie die Verwendung eines performanteren Analog-Digital Umsetzers wären obligatorisch. Die meisten Systeme sind jedoch nicht so flexibel anpassbar.

[1] Die Funktion lcm(V) liefert das "least common multiple" also das kleinste gemeinsame Vielfache der Menge V

Selbstähnlichkeit der Transformationsmatrizen

Weitere Untersuchungen an der **Cob**- bzw. **Sib**-Matrix ergaben, dass die Berechnung von $\mathbf{Cob} \cdot \mathbf{Cob}^T$ (und analog dazu auch die **Sib**-Matrix) eine Matrix mit den Fourier-Koeffizienten liefert. Das gilt jedoch nur bei der entsprechenden Wahl der Abtastwerte pro Zeitfenster in Anlehnung an den vorigen Abschnitt.

Betrachtet man diese Matrix-Multiplikation, so kann diese auch als diskrete Kreuzkorrelation (siehe Gleichung 4.66) der verschiedenen Funktionen miteinander für einen Verschiebungsfaktor von $v = 0$, für jede Zeile der Matrix interpretiert werden.
Die Annahmen die getroffen wurden, um auf die korrekte Anzahl der Abtastwerte zu schließen kann dadurch verifiziert werden. Zusätzlich wird die Idee der Kompensation von Fehlern unter Verwendung der eingesetzten Rechteckfunktionen bestätigt.

$$R_{xy}(v) = \sum_{n=0}^{N-1} x_n \cdot y_{n+v} \tag{4.66}$$

$$\frac{1}{N} \cdot \left(\mathbf{Cob} \cdot \mathbf{Cob}^T\right) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\frac{1}{3} & 0 & \frac{1}{5} & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -\frac{1}{3} \\ 0 & -\frac{1}{3} & 0 & 1 & 0 & -\frac{1}{7} & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{1}{5} & 0 & -\frac{1}{7} & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{3} & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.67}$$

$$\frac{1}{N} \cdot \left(\mathbf{Sib} \cdot \mathbf{Sib}^T\right) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{1}{3} & 0 & \frac{1}{5} & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \frac{1}{3} \\ 0 & \frac{1}{3} & 0 & 1 & 0 & \frac{1}{7} & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{1}{5} & 0 & \frac{1}{7} & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.68}$$

5. Verifikation der Anwendbarkeit des Algorithmus

Im vorhergehenden Kapitel wurde die alternative Transformation auf Basis von Rechtecksignalen entworfen. Es war jedoch ersichtlich, dass ein persistenter Restfehler bei der Spektralschätzung verbleibt. Die Abschätzung dieses Fehlers stellt ein Problem dar, da dieser direkt vom Eingangssignal abhängt. Damit das RAFT-Verfahren jedoch in der Praxis angewandt werden kann, muss gezeigt werden, dass sich dieser Fehler entsprechend der Vorhersagen verhält und dass der entstehende Fehler so gering ist, dass er für die praktische Anwendung an ABS-Sensorsignalen nur einen geringen Einfluss auf das Verzerrungsmaß ausübt.

Für diesen Zweck wird im Folgenden ein dreistufiger Benchmark durchgeführt, welcher im ersten Abschnitt randomisierte Daten als Referenzwerte verwendet, die jedoch im Spektrum bandbegrenzt werden. Der zweite Teil widmet sich dem Benchmarking anhand der aus der Signalsynthese gewonnenen Daten, welche im Rahmen der Abschlussarbeit von Zippel [35] entstanden sind.

Abschliessend wird noch eine Verifikation des Algorithmus mittels eines repräsentativen Messdatensatzes aus Messreihen des *Radmessplatzes 3* durchgeführt. Diese Verifikation soll die Anwendbarkeit an realen ABS-Sensorsignaldaten zeigen und zusätzlich einen Weg aufgezeigen wie eine hohe Dichte an Testsignalen für die Verifikation von neuen Algorithmen zu erhalten ist.

Für diese Benchmarks wurde MATLAB verwendet, um Monte-Carlo Simulationen durchzuführen. Dabei werden wiederholt Signale mit zufälligen Amplituden- und Phaseninformationen für die Harmonischen erzeugt und zu einem Zeitsignal synthetisiert. Anschließend wird sowohl eine DFT als auch eine RAFT durchgeführt und unter Verwendung von beiden Spektren jeweils das Verzerrungsmaß HD_6 berechnet. Aus diesen Messreihen werden dann Histogramme erstellt, die den Fehler, der durch die RAFT-Approximation entsteht, visualisieren sollen.

Sämtliche Berechnungen wurden mit Gleitkomma- und nicht mit Festkommadatentypen durchgeführt, da die Leistungsfähigkeit der sogenannten Fixed-Point Objekte für umfangreiche Simulationen als nicht ausreichend befunden wurde. Eine händische Festkomma-Implementation beider Algorithmen ist außerdem sehr aufwändig. Der RAFT-Algorithmus

wurde trotzdem in vereinfachter Festkomma Darstellung realisiert, um die Hardwareimplementierung zu verifizieren. Für die Monte-Carlo-Simulationen wurde diese Implementierung nicht genutzt.

In dieser Arbeit werden zwei Fehlermaße genutzt. Es muss eine differenzierte Betrachtung erfolgen einerseits muss die Fehlschätzung der Harmonischen betrachtet werden, andererseits die Auswirkungen die das neue Verfahren auf den HD_K hat. Die reine Betrachtung des Fehlers im Verzerrungsmaß ist nicht ausreichend, da das HD_K-Verzerrungsmaß nicht eindeutig ist.

Konstruieren wir beispielsweise ein Signal mit identischen Leistungen für die erste und zweite Harmonische und ohne weitere spektrale Anteile, so liefert dieses einen HD_6 Verzerrungswert von $\approx 70\%$. Wählen wir ein weiteres Signal mit identischen Leistungen für die erste und dritte Harmonische, wieder ohne weitere spektrale Anteile, so wäre der Verzerrungsmaßindikator HD_6 dieses Signals identisch mit dem vorigen.

Damit jedoch eine genauere Aussage über den Fehler der Spektralschätzung getroffen werden kann, wird zusätzlich der quadrierte Amplitudenfehler der einzelnen Harmonischen summiert, um die Verzerrungen genauer zu betrachten (siehe Gleichung 5.1).

In der Anwendung an ABS-Sensorsignalen ist dies jedoch zu vernachlässigen, da lediglich das Verzerrungsmaß ausgewertet wird. Einen repräsentativen Indikator für den Fehler in Bezug auf die Berechnung des Verzerrungsmaßes HD_6 erhalten wird, indem die absolute Differenz zwischen den DFT und den RAFT berechneten HD_6 Werten gebildet wird (siehe Gleichung 5.2).

Die Festlegung von sechs Spektrallinien für die Auswertung des Spektrums wurde gewählt, damit die maximale Anzahl an Harmonischen für die Anzahl der Abtastwerte von 240 analysiert werden kann (siehe Tabelle 4.2). Für die Simulationen werden die Amplituden der Harmonischen im Bereich $-1 \leq A \leq +1$ mit zufälligen Phasen im Bereich von $0 \leq \varphi < 2\pi$ gewählt.

Die verwendeten Fehlermaße
Zur Berechnung des quadratischen Fehlers der Amplitudenabweichung wird folgende Gleichung genutzt (K: Anzahl der Harmonischen):

$$e_{sq} = \sum_{k=1}^{K} \left(H_{DFT}(k) - H_{RAFT}(k)\right)^2 \tag{5.1}$$

Die Berechnung der Abweichung des Verzerrungsmaßes wird bestimmt über die folgende Gleichung:

$$e_{HD} = HD_{K,DFT} - HD_{K,RAFT} \tag{5.2}$$

5.1. Verifikation des Algorithmus mit Monte-Carlo-Simulationen

Für die erste Verifikation des Algorithmus wird eine Monte-Carlo-Simulation durchgeführt, die sich an die im Abschnitt 4.2 vorgeschlagenen Randbedingungen hält. Für diesen Zweck werden die zu analysierenden Signale mit zufälligen Amplituden- und Phaseninformationen in dem Bereich der zu untersuchenden Harmonischen synthetisiert. Durch diese Restriktion wird das Signal sehr stark bandbegrenzt.

Mit den gewählten 240 Abtastwerten pro Erfassungsfenster ist es maximal möglich sechs Harmonische mit einer guten Approximation zu erhalten, wie in Abschnitt 4.3.2 gezeigt wurde. Die Testsignale werden demnach nur aus verschiedenen Kombinationen dieser sechs Harmonischen synthetisiert und anschließend sowohl mit der DFT als auch mit der RAFT analysiert.

Das Resultat dieser Monte-Carlo-Simulation (siehe Abbildung 5.1) ergibt, dass der Fehler für diese Testsignale sehr klein ist, sowohl in Bezug auf den absoluten Fehler des Verzerrungsmaßes als auch auf das zweite gewählte Fehlermaß.

Den Rahmen der sehr stark bandbreitenbegrenzten Signale verlassend wurde im nächsten Schritt analysiert, wie sich die RAFT für weitere Testsignale verhält. Hierfür wurden Signale mit 31 zufälligen harmonischen Spektralkomponenten generiert und wiederum genau betrachtet. In Abbildung 5.2 ist deutlich ersichtlich, dass der Fehler im Verzerrungsmaß in einem realen System nicht mehr tolerierbar wäre, da Abweichungen von bis zu $\pm 60\%$ im Verzerrungsmaß HD_6 auftreten.

Das RAFT-Verfahren ist demnach nicht für beliebige Signale anwendbar. Es kann nicht für sämtliche Testsignale eine gute Spektrumsapproximation erhalten werden. Die bisherigen Untersuchungen an ABS-Sensorsignale in den vorigen Arbeiten (siehe Zippel [35], Jegenhorst [9]) ergaben, dass die Signale, die in einem solchen ABS-System auftreten, nicht vollkommen zufälliger Natur sind. Zur Überprüfung der Anwendbarkeit an diesen speziellen Signalen wurde eine Verifikation anhand der von Zippel [35] erstellten magnetostatischen Simulationen durchgeführt.

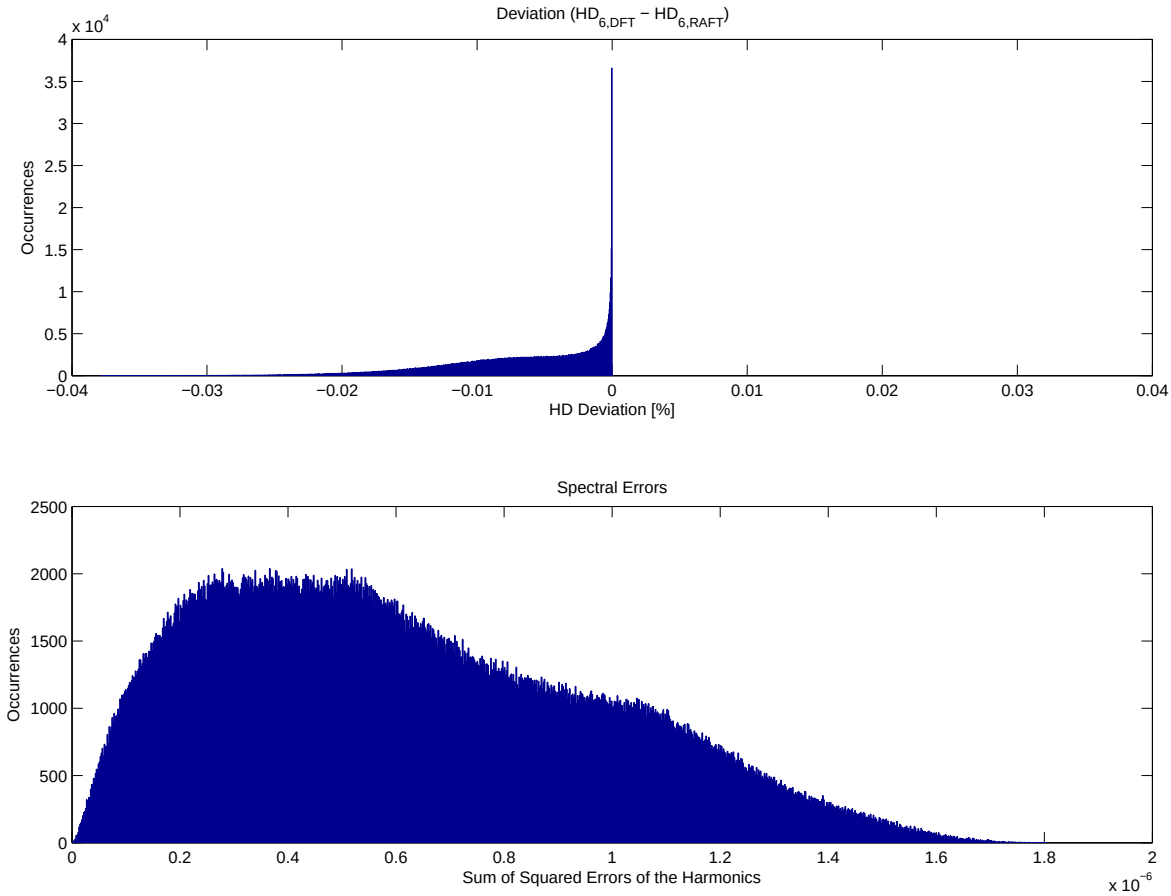


Abbildung 5.1.: Darstellung der Monte-Carlo-Simulation mit eingeschränkten Harmonischen in dem beschränkten Bereich der zu untersuchenden Harmonischen: $H_k = 0 \; \forall \; k > K$ (K: Anzahl der zu untersuchenden Harmonischen).

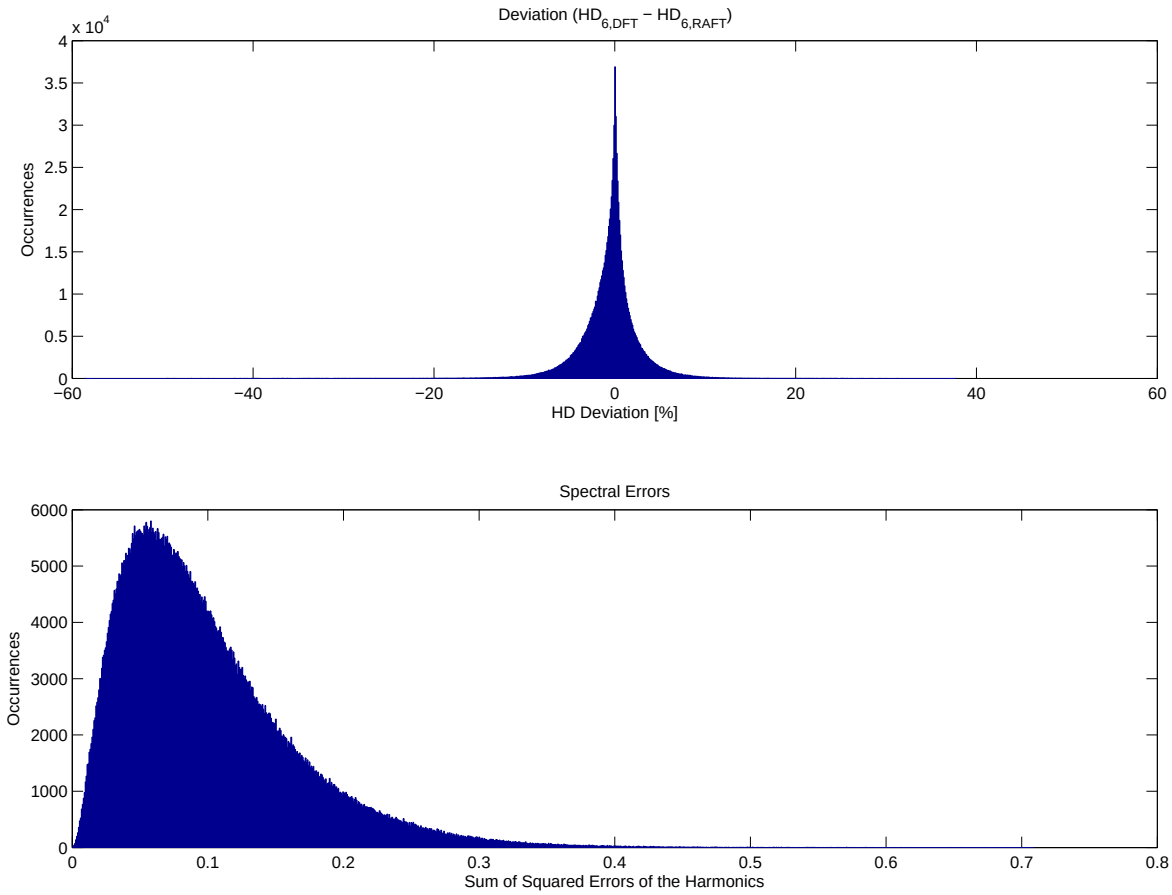


Abbildung 5.2.: Darstellung der Monte-Carlo-Simulation mit eingeschränkten Harmonischen in dem beschränkten Bereich der zu untersuchenden Harmonischen: $H_k = 0 \; \forall \; k > 31$ (K: Anzahl der zu untersuchenden Harmonischen). Die Anzahl der maximalen Harmonischen wurde durch die bisherige Hardwareimplementierung bestimmt, welche mit 64 Samples arbeitet, es ergeben sich unter diesen Voraussetzungen $\frac{N}{2} - 1 = 31$ mögliche gleichverteilte Harmonische.

5.2. Test des Algorithmus mit synthetischen Stimuli-Signalen aus der EM-Simulation

In der Arbeit von Zippel [35] wurden unter Verwendung von CST EM Studio magnetostatische Feldsimulationen des ABS-Sensorsystems durchgeführt. Aus den Ergebnissen dieser Simulationen konnte die Brückenspannung des ABS-Sensors synthetisiert werden. Es wurden insgesamt 31 verschiedene Signalformen durch diese Art der Signalsynthese generiert, die im Folgenden für eine erste Untersuchung an anwendungsbezogenen Signalen tatsächlich genutzt werden konnten.

Für die Darstellung dieser wenigen Signale ist ein Histogramm, welches in den vorigen Abschnitten genutzt wurde, um den auftretenden Fehler zu charakterisieren, nicht sinnvoll. Folglich wurde für die Darstellung des Fehlers ein alternativer Ansatz genutzt (siehe Abbildung 5.3).
Der durch die DFT berechnete HD_6-Indikator wird hierfür auf die Abszisse aufgetragen, während der RAFT-HD_6 auf der Ordinate des Koordinatensystems abgetragen wird. Zusätzlich wird die ideale Gerade und damit die 1:1 Zuordnung der beiden Verfahren hinterlegt. Eine weitere Fehlerkurve im unteren Teil der Abbildung verdeutlicht zudem den enstandenen absoluten Fehler quantitativ. Dabei wird es deutlich, dass der absolute Fehler einerseits nur für sehr hohe HD_6-Verzerrungsmaßwerte groß wird und andererseits, dass sich der Fehler nur in den Nachkommastellen des Verzerrungsmaßes bewegt. Daraus lässt sich auf eine erste Anwendbarkeit des Algorithmus schließen.

Das Resultat dieser Simulationen zeigt erstmals, dass das RAFT-Verfahren auch für ABS-Sensorsignale einsetzbar ist. In einem weiteren Test soll nun die Anwendbarkeit des RAFT-Verfahrens an realen Messdaten überprüft werden. Ein dafür eigens aufgenommener Messdatensatz wird im folgenden Abschnitt eingesetzt, um dies zu gewährleisten.

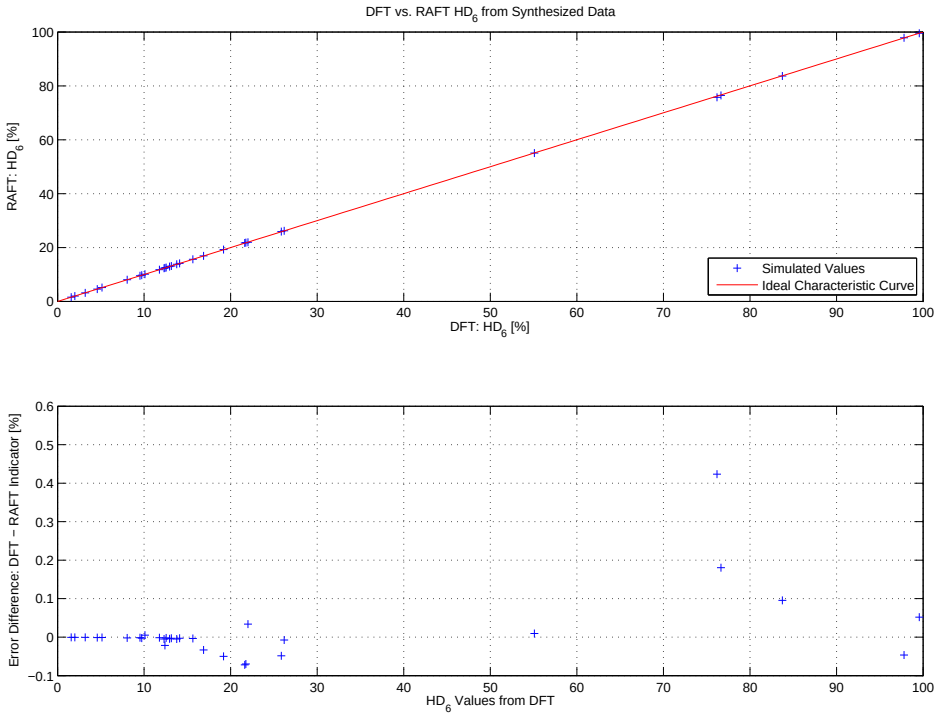


Abbildung 5.3.: Analyse des RAFT-Verfahrens in Bezug auf das Verzerrungsmaß HD_6, für diesen Zweck wird das Verzerrungsmaß mittels DFT und RAFT gebildet und einander gegenübergestellt. Die Fehlerkurve im unteren Abschnitt des Bildes zeigt nur minimale Absolute Abweichungen, welche tolerierbar sind für die Verwendung in einem realen ABS-Diagnose System.

5.3. Verifikation mit den aus Messdaten gewonnenen Signalen

Zum Erhalt eines möglichst großen und repräsentativen Messdatensatzes, wurde mit dem *Radmessplatz 3* (RMP3) eine sehr umfangreiche Messreihe aufgenommen. Zielführend war es möglichst viele verschiedene Verkippungen und Abstände mit dem Sensor anzufahren, damit die Messdaten das gesamte Spektrum an Möglichkeiten des Fehleinbaus abdecken können.
Der Radmessplatz erlaubt es insgesamt sechs Parameter in Minimalwert, Maximalwert und Inkrement zu definieren. Dabei handelt es sich um drei Abstandsachsen sowie drei Verkippungsachsen, die variabel und voneinander unabhängig parametrisiert werden können (siehe Abbildung 5.4).
Hierbei ist jedoch zu beachten, dass der Radmessplatz eine automatische Kollisionserkennung durchführt, um zu verhindern, dass der Sensor und das Encoderrad kollidieren. Dadurch können nicht alle gewählten Parameterkombinationen tatsächlich angefahren werden. Somit existieren für einige Parameterkombinationen keine auswertbaren Messdaten.
Für diese Messreihe wurde das axiale passive Referenzencoderrad DN5 verwendet. Die Parameter wurden gemäß Tabelle 5.1 gewählt. Die Erfassung diesen Datensatzes wurde in ungefähr acht Stunden mit dem *Radmessplatz 3* durchgeführt.

Parameter	Inkrement	min. Wert	max. Wert
x	0.0 mm	0.0 mm	0.0 mm
y	0.0 mm	0.0 mm	0.0 mm
z	−0.2 mm	0.0 mm	−2 mm
φ_x	5°	−15°	15°
φ_y	5°	−15°	15°
φ_z	15°	−45°	45°

Tabelle 5.1.: Auswahl der Parameter für die Erfassung des repräsentativen Messdatensatzes.

Mit den Werten aus der oben stehenden Tabelle ist es möglich 3773 (siehe Gleichung 5.3) voneinander verschiedene Kombinationen der Sensorpositionierung anzufahren.
Durch die bereits erwähnte Kollisionsabfrage des Messplatzes war es nur möglich ca. 1800 verschiedene Positionen des Sensors anzufahren. Dies stellt jedoch im Projekt ESZ-ABS einen der umfangreichsten Messdatensätze dar, welcher genutzt werden kann, um das Sensorsignal zu charakterisieren.

$$\begin{aligned} N_{\mathbf{total}} &= N_x \cdot N_y \cdot N_z \cdot N_{\varphi_x} \cdot N_{\varphi_y} \cdot N_{\varphi_z} \\ &= 3773 \end{aligned} \tag{5.3}$$

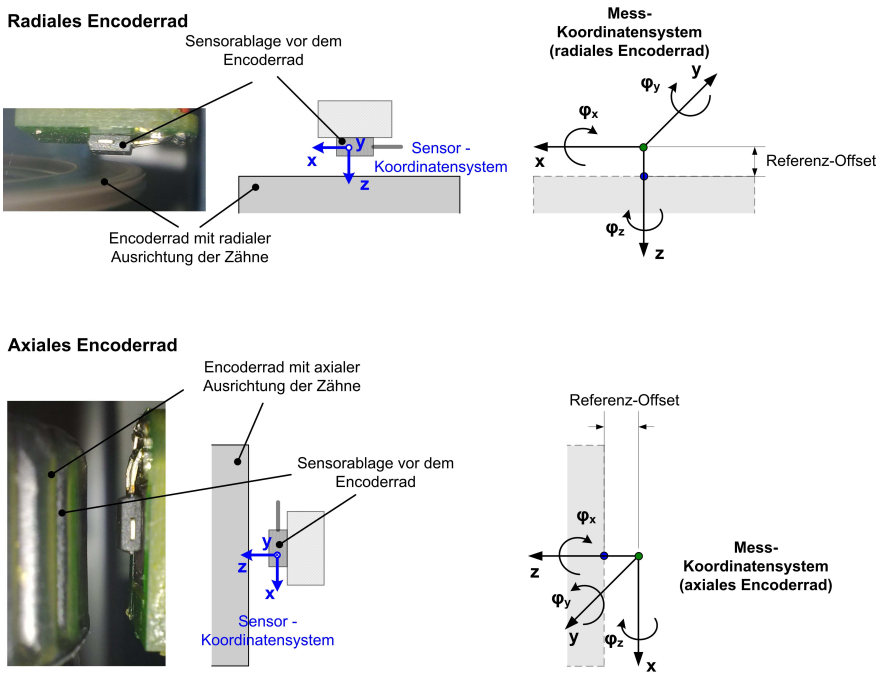


Abbildung 5.4.: Definition des Sensorkoordinatensystems: Radmessplatz 3. Für axiale und radiale Encoderräder nach Schörmer (entnommen aus [29]).

Ein Ziel dieser Messung war es eine Hüllkurve zu finden, die für die Monte-Carlo Simulation genutzt werden kann (siehe Abbildung 5.5). Mit dieser Hüllkurve können dann gleichverteilte Zufallssignale gewichtet werden, um die Algorithmen mit einem umfangreichen Spektrum an Eingangssignalen zu analysieren und dabei den Bezug zu den realen Daten zu erhalten.

Unter der Annahme, dass diese Hüllkurve sehr repräsentativ sei, wurde anschließend eine Monte-Carlo-Simulation durchgeführt, die in Abbildung 5.6 ersichtlich ist. Mit dieser Herangehensweise konnte die Dichte der Testsignale im Vergleich zu den bisherigen Ansätzen deutlich erhöht werden. Zusätzlich konnte die Geschwindigkeit zur Generierung von Testsignalen deutlich gesteigert werden (siehe Tabelle 5.2).

Durch den umfangreichen Datensatz war es nun auch möglich reale ABS-Sensorsignale mit dem neuen Verfahren zu betrachten, um wiederum die Anwendbarkeit des RAFT-Verfahrens an realen Signalen zu verifizieren. Die Resultate dieser Simulationen sind in Abbildung 5.7 und 5.8 dargestellt. Es wird deutlich, dass das RAFT-Verfahren tatsächlich für reale Messdaten anwendbar ist.

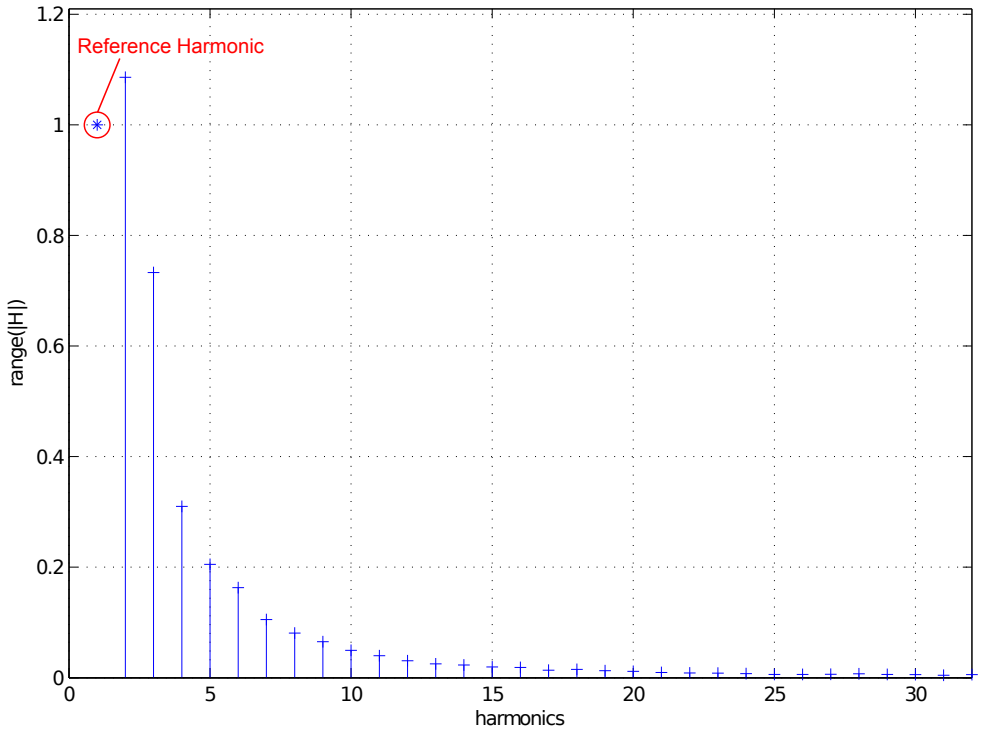


Abbildung 5.5.: Diese Abbildung repräsentiert den empirisch ermittelten Bereich in dem sich die absoluten Beträge der Harmonischen bewegen können. Dabei ist zu beachten, dass die gemessenen Signale jeweils auf die erste Harmonische und damit zur Zahnfrequenz normiert worden sind.

	Laufzeit [h]	**Anzahl der ermittelten Werte**
CST EM Studio	8	1
RMP3 Erfassung des Messdatensatzes	8	1 800
Zufallssignale gewichtet mit Hüllkurve	6	> 1 000 000

Tabelle 5.2.: Geschwindigkeitsvergleich zur Ermittlung von Testsignaldaten.

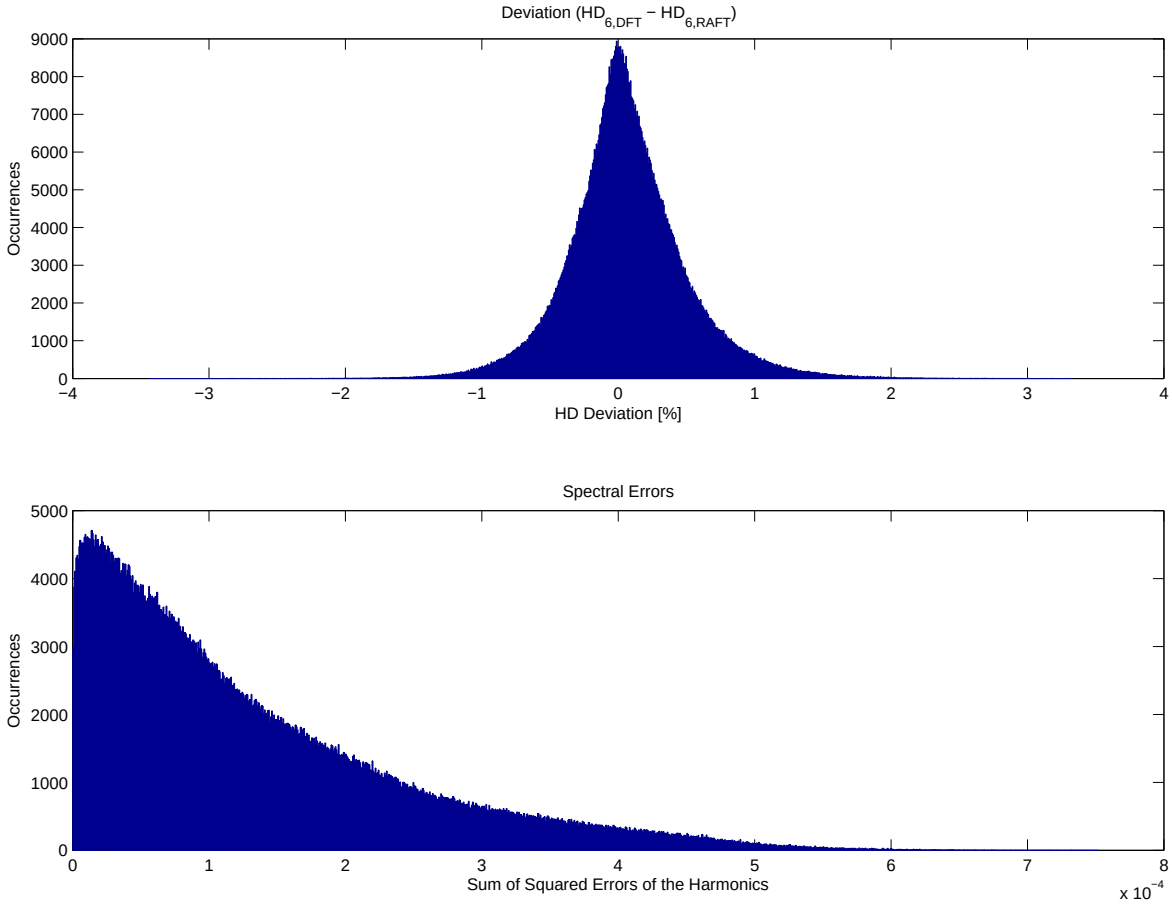


Abbildung 5.6.: Diese Darstellung zeigt die Monte-Carlo-Simulation unter Verwendung der ermittelten Hüllkurve. Durch diese Simulation kann sehr gut gezeigt werden, dass das entwickelte Verfahren durchaus anwendbar für diese Art der Signale ist, wenn ein Toleranzbereich von ± 4 % gefordert wird.

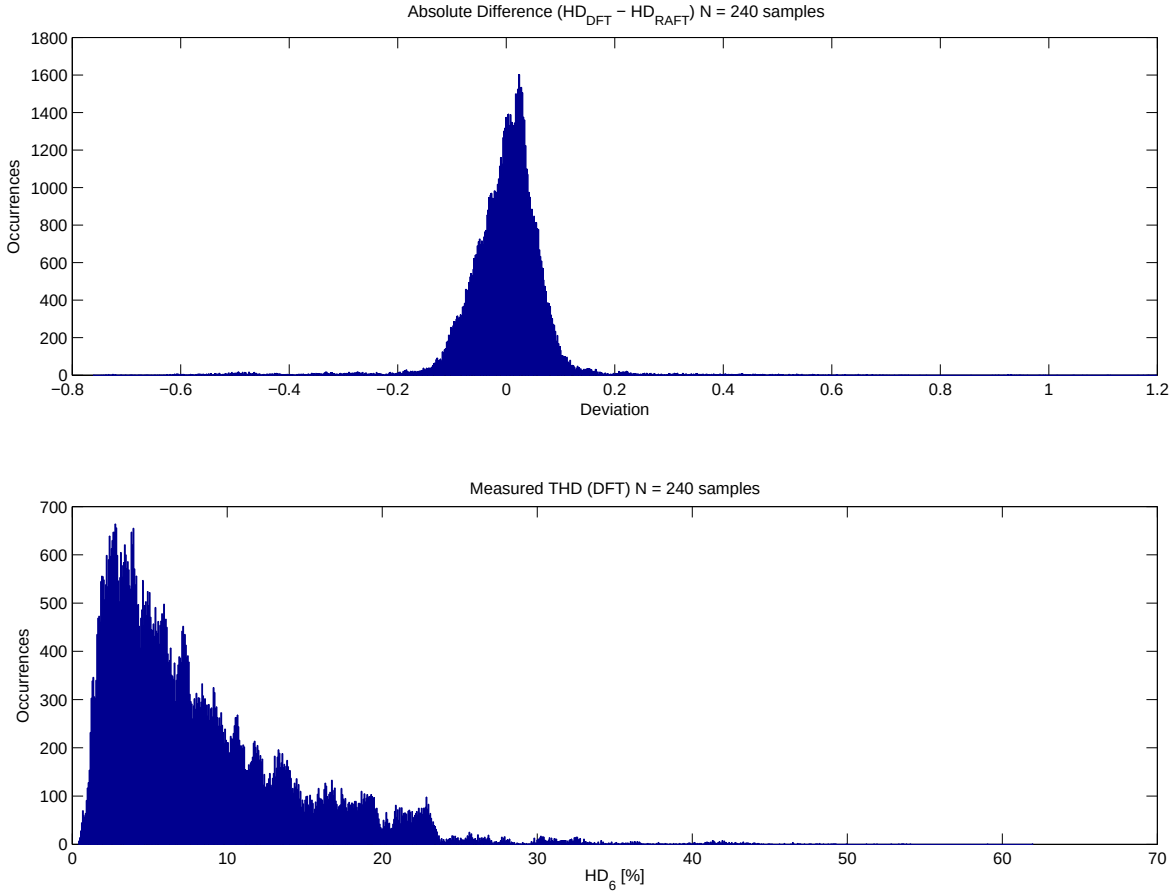


Abbildung 5.7.: Diese Abbildung zeigt die Abweichung, die das RAFT-Verfahren in Bezug auf den HD_6 besitzt. Die Eingangssignale stammen aus dem repräsentativen Messdatensatz. Im unteren Teil der Abbildung ist ein Histogramm über die im Messdatensatz vorhandenen HD_6-Werte dargestellt dargestellt.

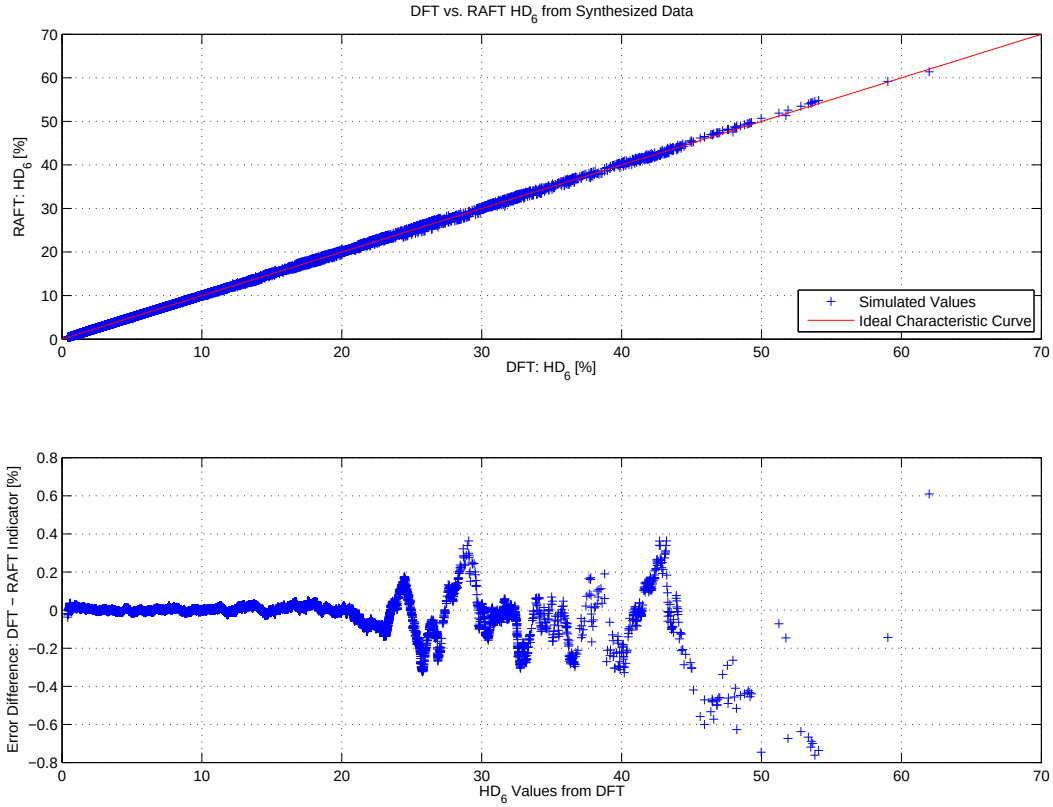


Abbildung 5.8.: Diese Abbildung zeigt die Auswertung der realen Messdaten in Bezug auf die Anwendbarkeit des RAFT-Verfahrens zur Berechnung des Verzerrungsmaßes HD_6. Im unteren Abschnitt der Abbildung ist deutlich erkennbar, dass sich der Fehler in einem Band von $\pm 0,8\%$ bewegt, was für die Anwendbarkeit in einem realen System als tolerabel anzusehen ist. Zusätzlich zeigt sich, dass der Fehler im Bereich bis zu 20% HD_6 deutlich kleiner als $0,1\%$ ist. Dies ist als gut zu bewerten, da die Signifikanz für große HD_6-Werte stark abnimmt.

5.4. Ergebnis der diversen Simulationen

Durch die diversen Simulationen konnte das RAFT-Verfahren sehr differenziert betrachtet werden. Im ersten Abschnitt zeigte sich, dass die RAFT nicht sehr gut für die Spektralschätzung beliebiger Signalformen geeignet ist. Begrenzte man das Signal sehr stark in seiner Bandbreite, so lieferte es eine sehr gute Approximation des DFT-Spektrums, sodass die in dem theoretischen Teil der Arbeit gewählten Annahmen so simulativ bestätigt werden konnten.

Erstmalig wurde es durch die Verifikation anhand der synthetischen Signale aus der Arbeit von Zippel [35] möglich eine Abschätzung zu erhalten, ob das RAFT-Verfahren in realen ABS-Sensorsystemen mit Selbstdiagnosefunktionalität eingesetzt werden kann. Mit dem sehr umfangreichen Messdatensatz konnte diese erste Annahme anschließend erfolgreich bestätigt werden. Es wurde gezeigt, dass sich der entstehende Fehler nur bei hohen Verzerrungsmaßwerten jenseits von 20% auswirkt, diese liegen jedoch außerhalb der sogenannten Safe-Operating Area und sind damit zu vernachlässigen. Außerdem zeigte sich, dass der absolute Fehler bei realen Signalen kleiner als 1% ist. Dies ist in einem realen System durchaus tolerierbar.

Mittels des repräsentativen Messdatensatzes konnte zudem eine Hüllkurve gefunden werden, um die ABS-Sensorsignaldaten zu charakterisieren. Dabei stellte sich heraus, dass die Hüllkurve nicht unmittelbar repräsentativ ist für alle ABS-Sensorsignale. Vielmehr hat es den Anschein, dass gewisse Kombinationen von Harmonischen in der Praxis nicht existieren. Aus Geschwindigkeitsaspekten betrachtet konnte durch diesen Ansatz jedoch die Leistungsfähigkeit der vorab durchgeführten Simulationen deutlich erhöht werden. Demnach wurde der Algorithmus unter Verwendung dieser Hüllkurve mit einer sehr hohen Abdeckungsdichte an Testsignale verifiziert. Es wurde eine deutlich höhere Diversität an Testsignalen durchgeführt, als tatsächlich benötigt wird. Demzufolge ist das gewählte Hüllkurven-Verfahren für eine erste Abschätzung der Anwendbarkeit von neuen Signalverarbeitungsalgorithmen in einem realen ABS-Sensorsystem sehr gut geeignet.

6. Exemplarische Implementierung

In diesem Kapitel wird auf die exemplarische Implementierung des RAFT-Verfahrens eingegangen. Es soll zeigen, dass es möglich ist diesen Algorithmus effizient in digitaler Hardware zu implementieren. Eine Proof of Concept Implementierung des RAFT-Verfahrens wird als Resultat erhalten. Der Systementwurf wird geschildert und die zur Implementierung notwendigen Parameter entsprechend zum Erhalt optimaler Resultate in Bezug auf Ressourceneinsparungen gewählt.

Abschließend wird die erstellte Implementierung kritisch mit der bisherigen DFT-Lösung verglichen. Die beiden Realisierungen werden sowohl in Bezug auf die benötigte Fläche als auch auf die Verarbeitungsgeschwindigkeit gegenübergestellt.

6.1. Systementwurf

Ein Top-down Ansatz wurde zum Entwurf des Systems gewählt. Das System wurde algorithmisch auf der obersten Ebene entworfen (siehe Abbildung 6.1) und dann heruntergebrochen, um kleinere Subsysteme zu definieren.

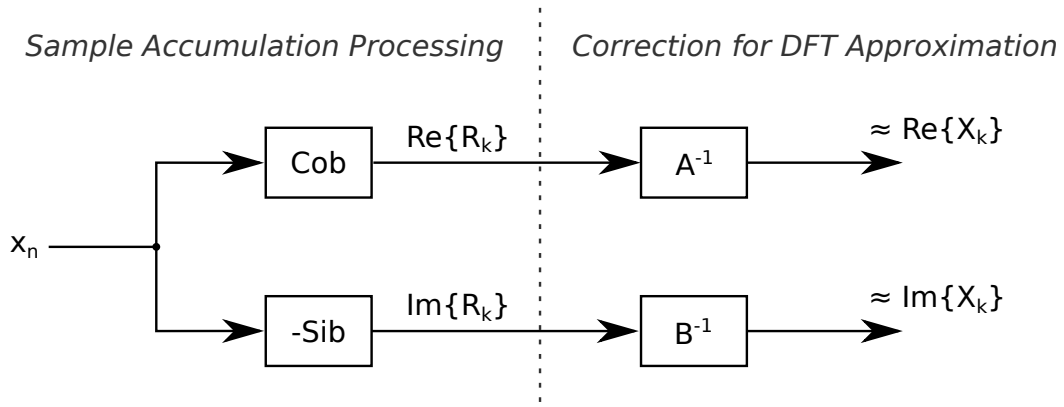


Abbildung 6.1.: Exemplarischer und stark vereinfachter Signalflussgraph der RAFT. Darstellung der Grundidee zur Implementierung.

Die RAFT-Implementierung wurde hierfür in zwei Hauptsysteme (das Pre- sowie das Postprocessing) gegliedert (siehe Abbildung 6.2). Die beiden Subsysteme konnten durch diesen

Ansatz mit den vorher definierten Schnittstellen getrennt voneinander entworfen und implementiert werden.

Das Preprocessing umfasst dabei die reine RAFT-Akkumulation, also die direkte Verwertung der Abtastwerte und damit die Berechnung aller benötigten R_k (siehe Kapitel 4). Das Postprocessing hingegen implementiert die Korrekturen, die nötig sind, um ein zur DFT ähnliches Spektrum zu erhalten. Es wird die Approximation von X_k sowie weitere Kalkulationen für die beiden Verzerrungsmaße HDI und HD_5 vorgenommen.

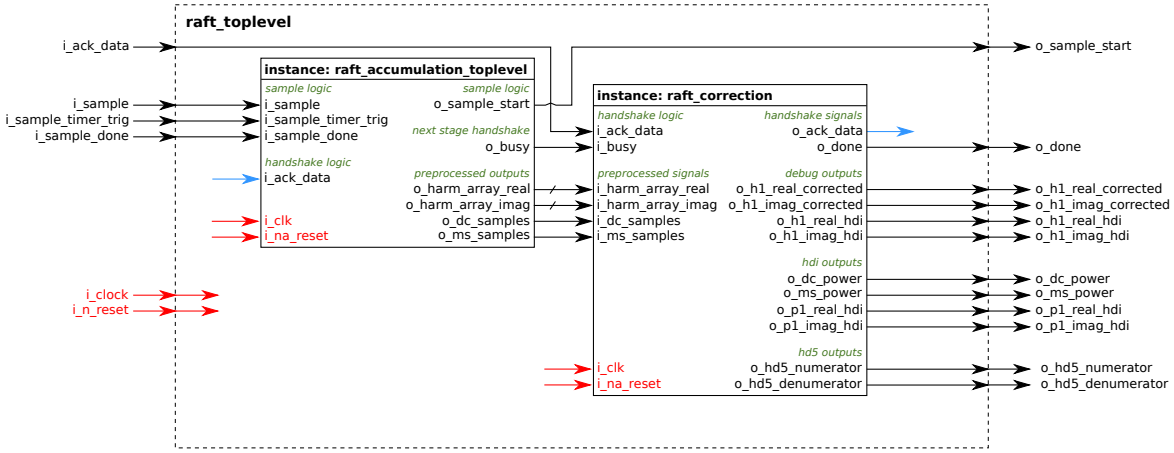


Abbildung 6.2.: Darstellung der RAFT-Implementierung in zwei Phasen: Preprocessing (links) und Postprocessing (rechts). Das Interface zur Aussenwelt wurde dabei der bisherigen Implementierung (siehe [27]) entnommen und entsprechend kompatibel designt.

6.2. Auswahl der Parameter für die Hardwareimplementierung der RAFT

Eine Hardware Implementierung erfordert die Bestimmung und Festlegung diverser Systemparameter. Dazu müssen die Randbedingungen des Systems analysiert und ausgewertet werden, um z.B. die Anzahl der Register klein zu halten. Beachtenswert sind die Faktoren aus den vorigen Kapiteln sowie weitere die in Hinblick auf eine CMOS-Chip Implementierung entstehen.

6.2.1. Systemparameter

Auf Basis der bisherigen Testchip Implementierung können einige Parameter fest gewählt werden. Im Projekt wurde eine maximale Systemtaktfrequenz von 16 MHz festgelegt, die als obere Systemgrenze anzusehen ist. Darüber hinaus wurde festgelegt, dass genau fünf Harmonischen zur Bestimmung des Verzerrungsmaßes genutzt werden sollen.

Desweiteren grenzt die Standardbibliothek des HIT-KIT [1] das System weiter ein, da nur ein Analog-Digital Umsetzer mit der Auflösung von maximal 10-Bit zur Verfügung steht.

Die Anzahl der Sample pro Periode wird basierend auf den Berechnungen aus Kapitel 4 auf 240 Samples pro Periode festgelegt. Die festen Parameter der Implementierung sind in Tabelle 6.1 aufgelistet.

Symbol	**Beschreibung**	**Wert**
N	Anzahl der Samples	240
K	Anzahl der zu untersuchenden Harmonischen	5
ADC_{res}	Auflösung des Analog-Digital Umsetzers	10 Bit
f_{max}	Maximale Systemtaktfrequenz	16 MHz

Tabelle 6.1.: Die festen Parameter der RAFT-Implementierung basierend auf den Resultaten der Analyse bzw. den Parametern des Testchips.

[1] Das HIT-KIT ist die von austriamicrosystems zur Verfügung gestellte Erweiterung der Cadence Toolchain für die ASIC-Implementierung

6.2.2. Designentscheidungen

Für die Umsetzung einer konkreten Hardwareimplementierung bestehen viele Möglichkeiten. Primär ist es notwendig sich vorher bewusst zu machen, welche Ressourcen eingesetzt werden sollen, um eine möglichst optimale anwendungsspezifische Lösung zu finden. Die Designentscheidungen werden dabei von zwei Zielen geprägt:

- Der neue Algorithmus soll in Zukunft in ABS-Sensoren mit Selbstdiagnosefunktionalität direkt im Mikrosystem des Sensors eingesetzt werden. Daher ist es das erste Ziel eine möglichst kleine Implementierung zu erhalten (in Bezug auf die benötigte Chip-Fläche).
- Die zweite Zielsetzung besteht darin, eine Implementierung zu finden, welche mit der bisherigen auf dem Testchip von Sabotta [27] vergleichbar ist. Dazu wird eine dieser Implementierung ähnliche angestrebt, welche sich an die meisten Randbedingungen des bisherigen Systems hält.

Viele Test-Implementierungen einzelner Komponenten unter Verwendung des Cadence ERC Compilers zeigten, dass zum Erhalt einer optimalen Implementierung besonderer Wert auf die Anzahl der Multiplizierer und die der Multiplexer gelegt werden muss. Zudem ist es sinnvoll für einen komplexen Algorithmus (die RAFT) eine sehr einfache Steuerungslogik zu implementieren. Dies kann die Suche nach Fehlern bzw. deren Korrektur deutlich vereinfachen.

Für die in dieser Arbeit vorgestellte Realisierung wurde eine hybride Implementierungsvariante gewählt. Sie benutzt lediglich einen Multiplizierer für die Berechnung der Korrekturen und die der quadratischen Leistungen. Es zeigte sich außerdem durch weitere Implementierungstests, dass es sich nicht lohnt Addierer und Subtrahierer mit allen Ressourcen zu teilen. Demzufolge wurde auch hier ein Mittelweg gewählt. Insgesamt werden fünf Addierer- und Subtrahierer genutzt, die zwischen der Verarbeitung von Real- und Imaginärteil umgeschaltet werden können, wobei die dafür benötigten Multiplexer kaum ins Gewicht fallen.

Dieses Design ermöglicht die Implementierung einer sehr einfachen Steuerungslogik in einem endlichen Zustandsautomaten. Der Vorteil dieser einfachen Steuerungslogik ist die sehr schnelle Verarbeitung eines Abtastwertes. Das große Defizit der erhöhten Anzahl an Abtastwerten kann durch diese Implementierung kompensiert werden.

Zur Vermeidung eines weiteren Multiplizierers im Systemdesign wurde zusätzlich ein Teil des Postprocessings für die Verzerrungsmaßberechnung vorverlagert. Die Berechnung der Leistungen der einzelnen Harmonischen und deren Summation wurde in das Postprocessing der RAFT integriert. So kann eine große Anzahl von Pipeline-Registern im System eingespart werden. Ebenso müssen weniger Multiplexer für die Harmonischen verwendet werden, da diese Signale bereits durch die spektrale Korrektur der RAFT mit dem Multiplizierer verbunden sind.

Die Entscheidungsmatrix für die Wahl der verschiedenen Parameter einer Implementierung ist in Tabelle 6.2 einzusehen.

Ressourcen	**Sequentiell**	**Parallel**	**Hybrid**	**Priorität**
Multiplizierer	+ +	- -	+	sehr hoch
Multiplexer	- -	+ +	-	hoch
Addierer	+ +	- -	+	niedrig
Subtrahierer	+ +	- -	+	niedrig
Steuerungslogik (FSM)	- -	+ +	+	niedrig
Verarbeitungszeit	- -	+ +	+	hoch

Tabelle 6.2.: Die Entscheidungsmatrix zur Bestimmung der Parameter für die Hardware Implementierung der RAFT. Die Priorität bezieht sich dabei auf die Flächeneffizienz der einzelnen Logikressourcen.

6.2.3. Bestimmung der Bitbreiten für die Akkumulationsregister

Die Bitbreiten für die Akkumulationsregister lassen sich relativ einfach bestimmen.
Es ist hinlänglich bekannt, dass die Addition zweier N-Bit breiter Vektoren maximal einen $N+1$ Bit breiten Ausgangsvektor liefert. Wieviele Bits für die gesamte Akkumulation der RAFT benötigt werden, kann man dementsprechend mit dem Logarithmus zur Basis zwei sehr einfach bestimmen (siehe Gleichung 6.1).

Zur Berechnung der Bitbreiten für die Akkumulationsregister der bisherigen Lösung kann man nun zusätzlich den diskreten Sinus miteinbeziehen. Die maximal benötigte Registerbitbreite ergibt sich durch die Multiplikation der Abtastwerte mit den LUT-Werten unter anschließender Akkumulation. Die Multiplikation eines N-Bit breiten Wertes mit einem M-Bit breiten Wert liefert einen $N+M$-Bit breiten Wert. Gewichten wir diesen Wert nun mit der diskreten Sinus-Funktion, so kann die exakte Bitbreite des Akkumulationsregisters bestimmt werden (siehe Gleichung 6.3). Diese Berechnung konnte durch empirische Simulationen verifiziert werden.

Die Gleichungen lassen erkennen, dass die RAFT deutlich weniger Bits pro Register im Vergleich zur DFT benötigt. Das ist bemerkenswert weil in die RAFT-Berechnung insgesamt 240 Abtastwerte eingehen, während die rDFT lediglich 64 Abtastwerte nutzt. Folglich erhalten wir durch die Anwendung des RAFT-Verfahrens eine Ersparnis von je 2x8 Bit für jede Harmonische. Der Faktor zwei ist hierbei bedingt durch die getrennte Verarbeitung nach Real- und Imaginärteil.

Die Registerbreiten für die Akkumulation können wie folgt bestimmt werden (N: Anzahl der Samples, ADC: Auflösung des Analog-Digital Umsetzers in Bit angegeben):

RAFT-Realisierung

$$\frac{Bit}{Register} = ceil\left(\log_2\left(N \cdot 2^{(\mathrm{ADC})}\right)\right) \tag{6.1}$$

$$= ceil\left(\log_2\left(240 \cdot 2^{10}\right)\right) \tag{6.2}$$

$$= 18$$

Alte Lösung (DFT)

$$\frac{Bit}{Register} = ceil\left(\log_2\left(\sum_{k=0}^{N-1} 2^{(\mathrm{LUT+ADC})} \cdot \left|sin\left(2\pi\frac{k}{N}\right)\right|\right)\right) \tag{6.3}$$

$$= ceil\left(\log_2\left(\sum_{k=0}^{63} 2^{(10+10)} \cdot \left|sin\left(2\pi\frac{k}{64}\right)\right|\right)\right) \tag{6.4}$$

$$= 26$$

6.3. Blockdiagramme der Signalverarbeitung

Die Visualisierung mittels Blockdiagrammen ist hilfreich zur Darstellung und Implementierung der Signalverarbeitung. Insgesamt werden zwei Indikatoren für das Verzerrungsmaß (HD_5 und HDI) vom System berechnet. Dementsprechend existieren zwei verschiedene Signalpfade, die jeweils in einem Indikator für das Verzerrungsmaß enden.

Die Abbildung 6.3 beschreibt die Berechnung des HD_5 Indikators vereinfacht. Die Samples werden für diesen Zweck erfasst und auf die Akkumulationsregister addiert bzw. von diesen subtrahiert. Nach Erfassung des letzten Samples findet die spektrale Korrektur statt, um ein zur DFT ähnliches Spektrum zu erhalten. Im letzten Schritt werden nun die Harmonischen quadriert und summiert, um den Zähler und den Nenner für die Berechnung des HD_5 zu bilden (siehe Gleichung 6.5).

$$HD_{5,\%} = 100 \cdot \sqrt{\frac{Numerator}{Denumerator}} \quad \% \tag{6.5}$$

$$= 100 \cdot \sqrt{\frac{\sum_{i=2}^{5} |H_i|^2}{\sum_{i=1}^{5} |H_i|^2}} \ \% \tag{6.6}$$

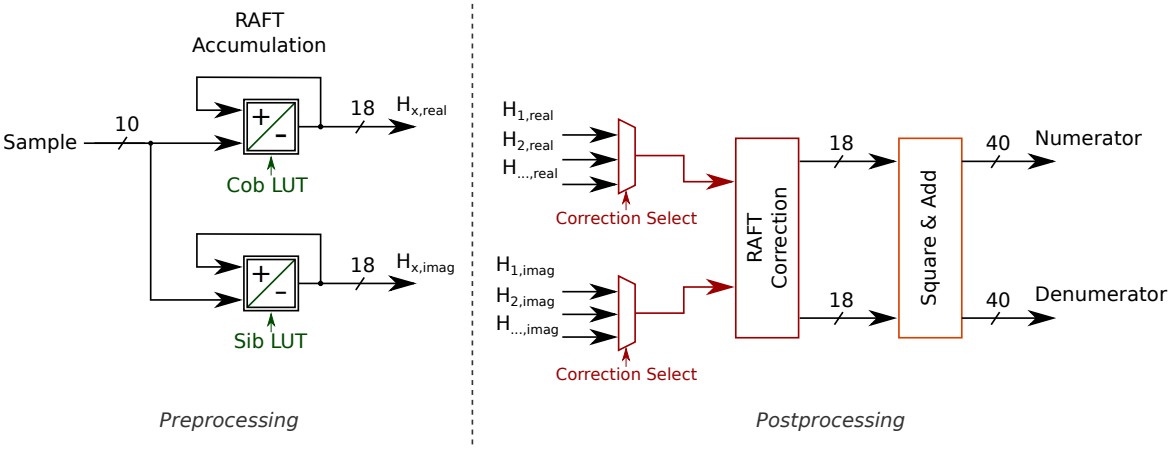


Abbildung 6.3.: Diese Abbildung zeigt die schematische Berechnung einer Harmonischen. Im Preprocessing erfolgt die Akkumulation der Abtastwerte, während im Postprocessing die Harmonischen ggf. korrigiert und dann für das Verzerrungsmaß akkumuliert werden.

Die Berechnung des *HDI* ist in Abbildung 6.4 ersichtlich. Durch die konsequente Multiplikation aller benötigten Werte mit dem Faktor N^2 kann eine sehr effiziente Berechnung der Harmonischen auch ohne Divison realisiert werden. Hierfür muss lediglich eine zusätzliche Konstantenmultiplikation durchgeführt werden. Bei Anwendung der RAFT sollten keine Skalierungen mit dem Faktor 1/N eingesetzt werden, um Divisionsoperationen zu vermeiden (bzw. Multiplikation mit: $1/240$).
Zum Erreichen dieses Ziels muss die Berechnung des Verzerrungsmaßes angepasst werden. Hierfür wird die bereits erwähnte Korrektur aller Leistungen mit dem Faktor N^2 eingesetzt. Multipliziert man diesen Faktor an die Berechnung aller Leistungen heran, so kann man in Gleichung 6.8 sehen, dass lediglich eine Konstantenmultiplikation in der Realisierung hinzugefügt werden muss.

Für die Berechnung der Leistung der Bezugsharmonischen (P_{mh}) muss noch eine weitere Korrektur mit den Faktor $\pi/4$ erfolgen. Dieser Faktor ergibt sich direkt aus der Fourier-Reihenentwicklung (siehe Abschnitt 4.1.2).

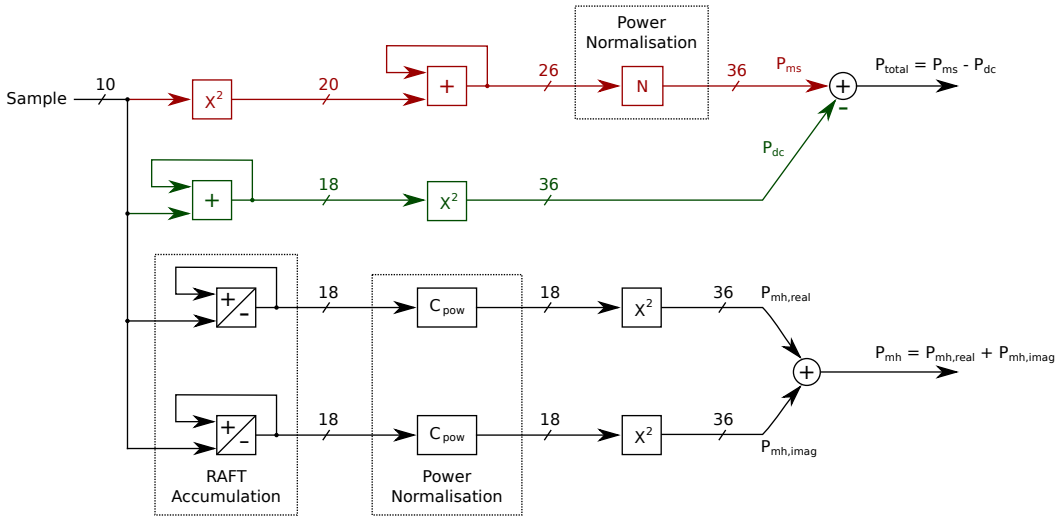


Abbildung 6.4.: In dieser Abbildung ist der exemplarische Signalfluss für die HDI-Berechnung unter Verwendung des RAFT-Verfahrens dargestellt.

$$P_{DC} = \left(\frac{1}{N} \cdot \sum_{n=0}^{N-1} x_n \right)^2 \quad \rightarrow \quad N^2 \cdot P_{DC} = \left(\sum_{n=0}^{N-1} x_n \right)^2 \tag{6.7}$$

$$P_{MS} = \frac{1}{N} \left(\cdot \sum_{n=0}^{N-1} x_n^2 \right) \quad \rightarrow \quad N^2 \cdot P_{MS} = N \cdot \left(\sum_{n=0}^{N-1} x_n^2 \right) \tag{6.8}$$

$$P_{mh} = 2 \cdot \left| \frac{\pi}{4N} \cdot H_{mh} \right|^2 \quad \rightarrow \quad N^2 \cdot P_{mh} = 2 \cdot \left| \frac{\pi}{4} \cdot H_{mh} \right|^2 \tag{6.9}$$

6.4. Interpolation

Durch das Vorhandensein der 240 Samples pro Abtastzeitfenster ist es nötig, dass das Eingangssignal in der Abtastrate umgesetzt werden muss. Der eingesetzte Analog-Digital Umsetzer, der aus dem HIT-KIT für die ASIC Implementierung stammt, wird diesen hohen Anforderungen nicht mehr gerecht. Das ABS-Sensorsignal kann zwischen 1 und 2500 Hz variieren, dies erfordert einen ADC mit 600k Samples/s. Der uns zur Verfügung stehende Analog-Digital Umsetzer ist jedoch auf 100k Samples/s limitiert. Zur Lösung dieses Problems wurde eine aufwandsreduzierte Abtastratenumsetzung implementiert. Hierfür werden 60 Samples pro Abtastzeitfenster in 240 Samples umsetzt. Für diesen Zweck wird das Signal in zwei Stufen jeweils um den Faktor 2 heraufgesetzt und linear interpoliert.

Diese Art der Interpolation ermöglicht ein sehr effizientes Hardware-Design für die Abtastratenumsetzung des Eingangssignales. Insbesondere profitiert die Tiefpassfilterung von diesem Ansatz. Es müssen nur sehr einfache Mittelwertfilter eingesetzt werden, die lediglich Multiplikationen mit dem Faktor 1/2 verwenden (siehe Abbildung 6.5).
Diese Multiplikationen lassen sich sehr effizient in digitaler Hardware durch Schiebeoperationen bzw. Bitselektionen umsetzen. Es werden keine weiteren kombinatorischen Multiplizierer benötigt.

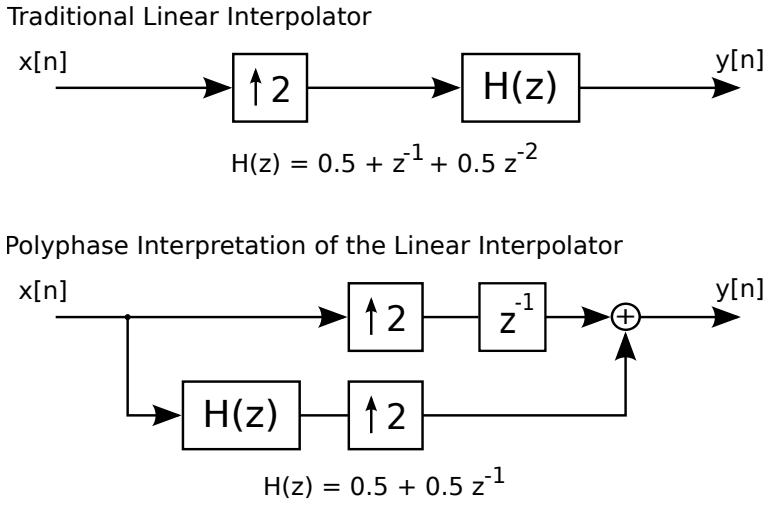


Abbildung 6.5.: Signalflussgraph der aufwandsreduzierten linearen Interpolation um den Faktor zwei in normaler sowie in Polyphasendarstellung. Die Polyphasenvariante wurde in VHDL implementiert, da diese eine effizientere Umsetzung ermöglichte.

6.5. Vergleich der Implementierungen (rDFT und RAFT)

Die von Sabotta [27] erstellte CMOS-Referenzimplementierung bildet die Grundlage für die weiteren Untersuchungen zur Bestimmung der benötigten Fläche auf dem Chip. Dazu wurde, unter Verwendung des Cadence Encouter RTL Compilers, die digitale Hardware in Form von VHDL-Beschreibungen in Standardzellen synthetisiert. Aus diesen Syntheseresultaten lässt sich anschließend die Fläche extrahieren, die von diesen Standardzellen eingenommen wird.

Da die entwickelten Architekturen jedoch voneinander differieren, kann kein direkter Vergleich unternommen werden. Wie bereits im vorhergehenden Kapitel erwähnt wurde, ist die Berechnung von Teilen der Verzerrungsmaße sehr stark mit dem Postprocessing der RAFT-Implementierung verknüpft worden, um eine möglichst gute Nutzung der komplexen Ressourcen zu gewährleisten.
In der bisherigen Implementierung hingegen sind das Pre- und Postprocessing durch Pipeline-Register voneinander getrennt worden.

Anschließend wird noch ein Geschwindigkeitsvergleich vorgenommen, indem die beiden Implementierungen kritisch einander gegenüber gestellt und bewertet werden. Dabei spielt besonders die Verarbeitungsgeschwindigkeit eines Samples sowie die der abschließenden Korrektur eine große Rolle.

6.5.1. Ressourcenvergleich

Wie im vorigen Abschnitt beschrieben, kann kein direkter Vergleich zwischen den beiden Implementierungen durchgeführt werden, da diese sehr unterschiedlich aufgebaut sind. Trotzdem kann man zum Erhalt eines aussagekräftigen Resultates die bisherige Implementierung sehr genau betrachten und einzelne Instanzen herausgreifen. Zwar gehören sie nicht unbedingt unmittelbar zur Implementierung des Algorithmus, so wurden sie jedoch in das RAFT-Postprocessing integriert, um Ressourcen zu sparen. Dies führt nun dazu, dass ein Vergleich beider Implementierungen durchgeführt werden kann.

Bei einer derartigen Auswertung kann man eine ungefähre Abschätzung des Aufwands in Bezug auf die Chip-Fläche erhalten. Das Resultat dieser Auswertung ist in Tabelle 6.3 ersichtlich.

	benötigte Fläche [mm^2]	Prozentualer Flächenbedarf
DFT-Implementierung	0,567	100,00 %
RAFT-Verfahren	0,480	84,70 %
RAFT-Verfahren mit Interpolation	0,518	91,26 %

Tabelle 6.3.: Übersicht der benötigten Fläche der beiden implementierten Verfahren.

Bezogen auf die bisherige Implementierung benötigt das RAFT-Verfahren lediglich 84,7 % der Fläche. Verwendet man zusätzlich den aufwandsminimierten Interpolator, so benötigt die RAFT-Implementierung 91,26 % der bisher genutzten Fläche. Somit ist der Implementierung der RAFT in dem angestrebten System gegenüber der Implementierung der rDFT in Bezug auf die Chip-Fläche der Vorzug zu geben.

In den Abbildungen 6.6 und 6.7 ist die Fläche die beide Implementierungen benötigen, bezogen auf die einzelnen Verarbeitungseinheiten, dargestellt.

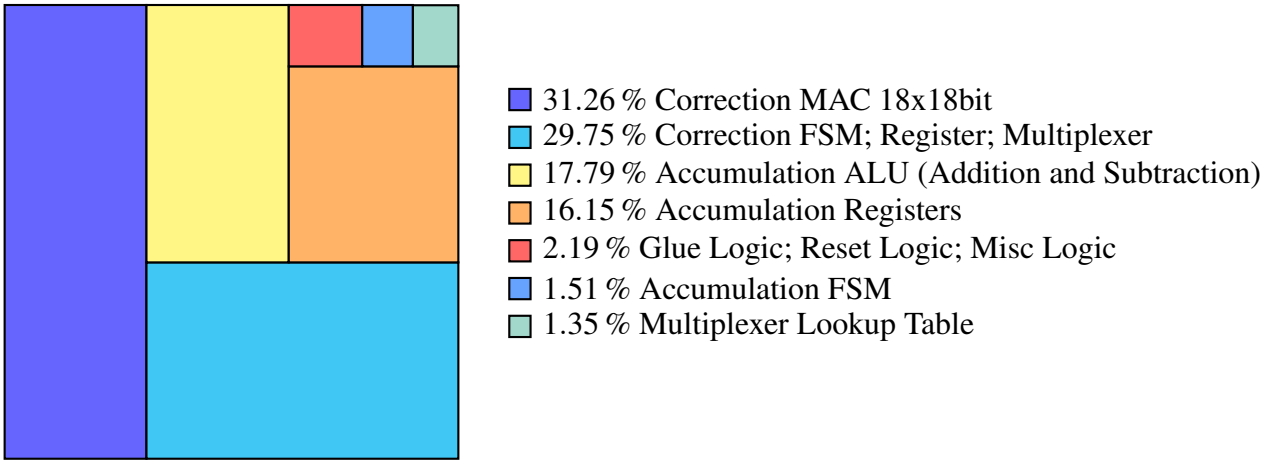


Abbildung 6.6.: Detaillierte prozentuale Darstellung der RAFT Implementierung.

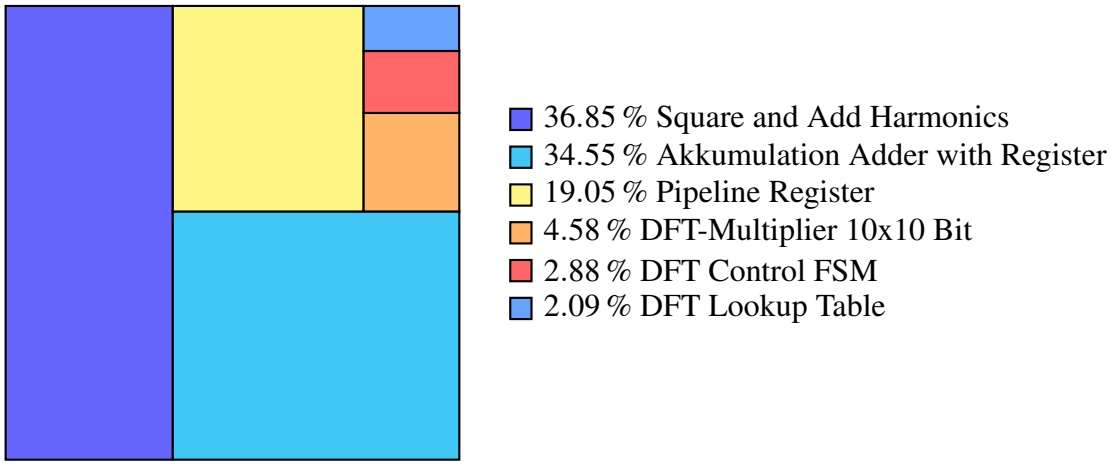


Abbildung 6.7.: Darstellung des Flächenbedarfs der aktuellen DFT-Implementierung.

6.5.2. Detaillierte Flächenbelegung der RAFT

Die Betrachtung der RAFT-Implementierung in Bezug auf die benötigte Fläche liefert sehr interessante Ergebnisse. Deutlich erkennbar ist, dass das Preprocessing, also die reine gesteuerte Akkumulation der Samples mittels Addierern und Subtrahierern, einen geringen Teil der gesamten Fläche benötigt. Lediglich 38,99% werden dafür eingenommen (siehe Abbildung 6.8). Da das Postprocessing sehr stark ausgedehnt wurde und nun einen großen Teil der Vorberechnungen für die beiden Verzerrungsmaße integriert, stellt dieses den größeren Anteil der beiden Teilsysteme dar.

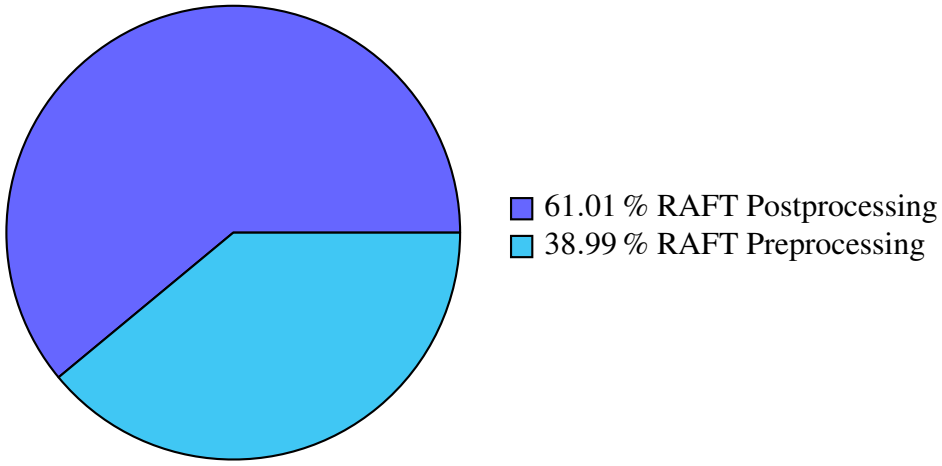


Abbildung 6.8.: Prozentuale Auswertung des Flächenbedarfs der RAFT Implementierung

Außerdem ist es sehr interessant, dass sich die Verteilung der benötigten Ressourcen auf die Domänen sequentieller und kombinatorischer Logik sowie Inverter und Buffer, in Bezug auf die DFT-Referenzimplementierung kaum verändert hat. Prozentual werden jeweils ungefähr gleich viele Ressourcen des entsprechenden Typs genutzt (siehe Abbildung 6.9 und Vergleiche mit 6.10).

Die eingesparten Pipeline-Register machen sich jedoch deutlich in der neuen Implementierung bemerkbar. Vermutlich sind sie hauptverantwortlich für die große Einsparung, die durch das RAFT-Verfahren erreicht werden konnte. Ebenfalls konnte auch ein sequentieller Multiplizierer eingespart werden. Dieser ist in der rDFT-Implementierung für das Quadrieren der Harmonischen zuständig und benötigt eine sehr große Anzahl an Flip-Flops. In der RAFT-Implementierung konnte dies durch den für die spektrale Korrektur eingesetzen kombinatorischen Multiplizierer realisiert werden. Dies hatte zur Folge, dass weitere Register-Standardzellen eingespart werden konnten.

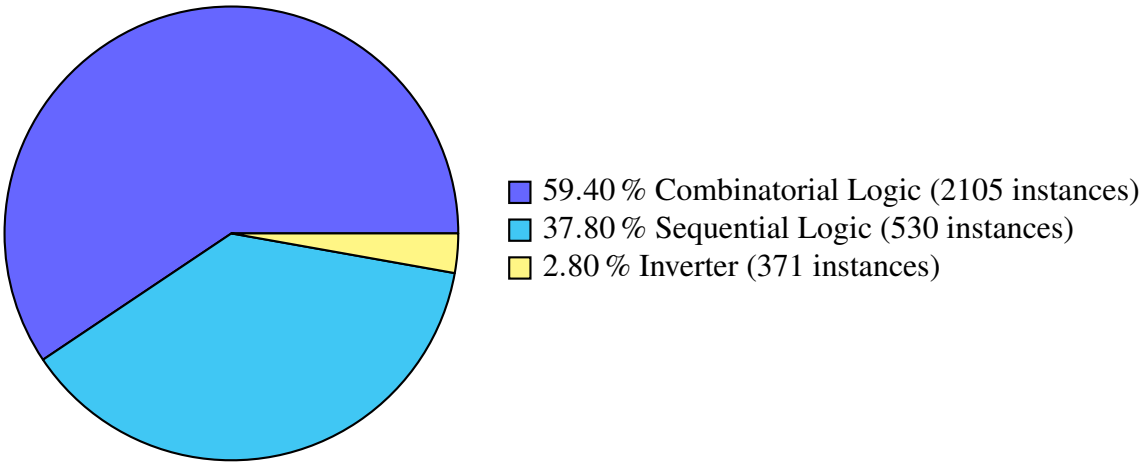


Abbildung 6.9.: Prozentualer Flächenbedarf der RAFT Implementierung in Bezug auf die verschiedenen genutzten Logiktypen

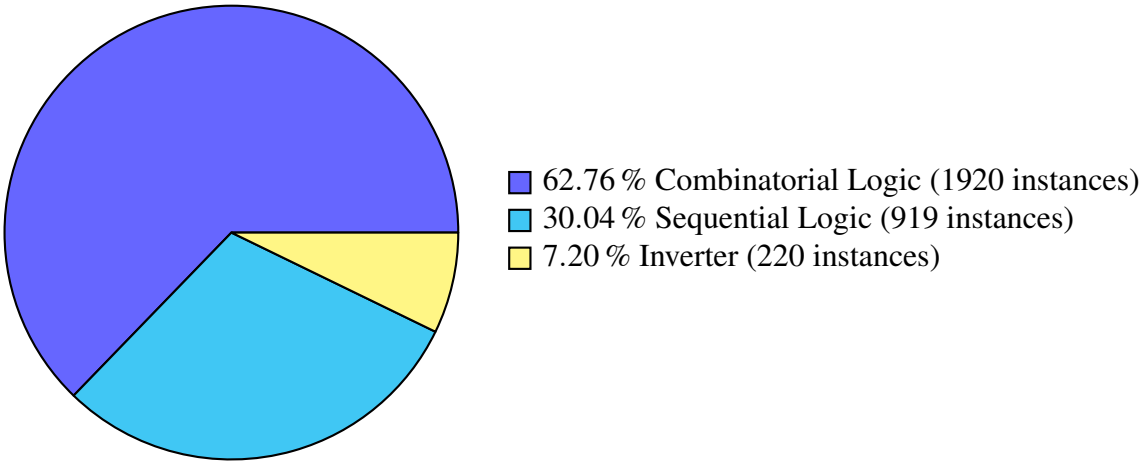


Abbildung 6.10.: Prozentualer Flächenbedarf der bisherigen DFT-Implementierung in Bezug auf die verschiedenen genutzten Logiktypen

6.5.3. Geschwindigkeitsvergleich

Für einen repräsentativen Geschwindigkeitsvergleich der beiden Implementierungen, müssen diese analysiert werden. Den wohl besten Vergleich liefert die Analyse der benötigten Taktzyklen in Bezug auf die einzelnen Verarbeitungsschritte. Als Referenz dient die RAFT-Implementierung, die einen großen Teil des Postprocessings (die Vorberechnung für die Verzerrungsmaße) inkludiert. Die Berechnung der rDFT bis hin zum Postprocessing anhand der bisherigen Realisierung kann momentan lediglich abgeschätzt werden, da diese sehr komplex implementiert wurde.

Die Verarbeitung der Daten beider Verfahrens kann in drei Verarbeitungseinheiten aufgeteilt werden:

- Die Verarbeitung eines Samples nach Erfassung durch den Analog-Digital-Umsetzer,
- die Korrektur des Ergebnisvektors, welcher von der RAFT für die Approximation des DFT-Spektrums genutzt wird
- und das Quadrieren und Akkumulieren der Harmonischen für die Vorberechnungen der Verzerrungsmaße.

Verarbeitungseinheit	**Referenz-Implementierung** (Taktzyklen)	**Neuer Ansatz (RAFT)** (Taktzyklen)
Prozessierung eines Samples	≈ 16	4
Korrektur des Ergebnisvektors	-	6
Quadrieren & Addieren der Harmonischen	≈ 240	15

Tabelle 6.4.: Tabellarische Übersicht der benötigten Taktzyklen der bisherigen Referenzimplementierung (rDFT) und des neuen Ansatzes (RAFT).

Die Tabelle 6.4 ist in Bezug auf die Implementierungen nicht sehr aussagekräftig, da diese mit einer unterschiedlichen Anzahl an Abtastwerten arbeiten (rDFT: 64, RAFT: 240). Formuliert man jedoch aus diesen Werten sehr einfache Gleichungen zur Approximation der benötigten Systemtaktzyklen (siehe Gleichungen 6.10 und 6.12), kann ein direkter Vergleich der beiden Verfahren durchgeführt werden.

$$\text{Cycles}_{\text{old}} \approx (N_{\text{DFT}}) \cdot 16 + 240 \tag{6.10}$$
$$\approx 1264 \tag{6.11}$$

$$\text{Cycles}_{\text{new}} \approx (N_{\text{RAFT}}) \cdot 4 + 6 + 15 \tag{6.12}$$
$$\approx 981 \tag{6.13}$$

Wie sich jetzt aus Gleichungen 6.11 und 6.13 erkennen lässt, ist die RAFT-Implementierung der bisherigen Referenz-Implementierung bezüglich der Verarbeitungsgeschwindigkeit überlegen. Durch diese Implementierung wird eine Reduzierung der benötigten Taktzyklen von $\approx 22,39\,\%$ erreicht.

6.5.4. Limitierungen

Wie bereits in den vorigen Abschnitten erwähnt, ist eine RAFT-Implementierung nicht immer direkt umsetzbar, um eine N-Punkte DFT zu ersetzen. Für die meisten Fälle ist es notwendig die Anzahl der Abtastwerte so zu erhöhen, dass die Möglichkeit zur Korrektur des RAFT-Ergebnisvektors gegeben ist und damit eine gute DFT-Approximation erreicht werden kann.

Es liegen weitere Limitierungen für diese Implementierung vor, weil die Anforderungen an die Signalverarbeitungen im System durch die Erhöhung der Sampleanzahl deutlich steigen. Die üblichen Skalierungen um Faktoren wie N oder $1/N$, bei denen N eine Potenz von zwei ist, sind zum Beispiel in digitaler Hardware sehr einfach realisierbar. Da jedoch die Anzahl der Abtastwerte N für RAFT-Implementierungen meistens keine Potenz von zwei ist (siehe Abschnitt 4.3.2), benötigt man eine andere Herangehensweise, um dies zu kompensieren.

Für die reine Spektralschätzung kann man diesen Skalierungsfaktor einfach aussparen und mit den daraus resultierenden größeren Bitbreiten arbeiten. Das führt wiederum einen höheren Ressourcenbedarf mit sich. Für einige Anwendungen kann man jedoch auch mit einem Faktor der Form $2^{(+/-)\,x}$ skalieren, falls der genaue Wert nicht von Bedeutung ist und nur das Verhältnis der Spektrallinien zueinander benötigt wird.

Außerdem stellte es sich heraus, dass es durchaus Systeme gibt, bei denen die Anzahl der Abtastwerte nicht einfach beliebig heraufgesetzt werden kann. Hinzukommend schränken weitere Faktoren das System ein, z.B. in der maximalen Taktfrequenz f_{max} des Analog-Digital Umsetzers oder dem maximalen Systemtakt.

7. Fazit und Ausblick

Diese Arbeit zeigte, dass die Konstruktion eines Verfahrens zur Spektralschätzung für ABS-Sensorsignaldaten auf der Basis von Rechteckfunktionen gelingt. In der Durchführung ergaben sich einige Limitierungen, die die allgemeine Anwendbarkeit des Verfahrens für beliebige Signale einschränken. Es stellte sich heraus, dass das Verfahren für den spezifischen Anwendungszweck an den ABS-Sensorsignalen sehr gut geeignet ist, da der entstehende Fehler in der Spektralschätzung für diese Signale äußerst gering ist.

Umfangreiche Simulationen und Tests konnten die Annahme der Anwendbarkeit bestätigen. Durch die Entwicklung eines Simulationsverfahrens unter Verwendung der anwendungsspezifischen Hüllkurve steht auch für weitere Arbeiten ein gutes Werkzeug zur Verfügung, um sehr gute erste Simulationen mit diesen Testsignalen durchzuführen.

Die exemplarische Implementierung zeigte deutlich, dass das Verfahren durch die geschickte Wahl von Systemparametern so effizient implementiert werden konnte, dass sowohl die Verarbeitungsgeschwindigkeit gesteigert werden konnte als auch die benötigte Fläche in einer ASIC-Implementierung geringfügig veringert werden konnte. Dies gelang trotz des Nachteils der viermal höheren Anzahl an Abtastwerten pro Erfassungszeitfenster.

Sollte jedoch in Zukunft die Entscheidung getroffen werden, nur den *HDI* als Indikator für das Verzerrungsmaß zu verwenden, wäre eine Implementierung in Form der reduzierten DFT oder in Form des Goertzel-Algorithmus wesentlich effektiver. Bei Anwendung der RAFT müssen immer mehrere Spektrallinien berechnet werden, um Korrekturen durchführen zu können. Für den Einsatz zur Bestimmung des HD_5-Verzerrungsmaßes zeigte sich hingegen, dass die RAFT-Implementierung deutlich effizienter als die der rDFT ist.

7.1. Ausblick

Für weitere Arbeiten stellten sich im Laufe der Erstellung dieser Ausarbeitung einige interessante Fragen, die leider nicht mehr beantwortet werden konnten. Deshalb wird im Folgenden auf einige ausgewählte Themen eingegangen, die der Autor für besonders untersuchenswert hält.

Goertzel-Algorithmus
Interessant wäre die Überprüfung der Reduktion von Ressourcen durch Implementierung des Görtzel-Algorithmus (siehe Abschnitt 3.4) anstelle des hier verwendeten Verfahrens (der RAFT) bzw. des alten Verfahrens (rDFT). Der Görtzel-Algorithmus ist ein sehr einfaches Verfahren, um einige wenige Spektralanteile zu berechnen. Im Hinblick auf die benötigten Multiplikationen scheint er bisher der effizienteste Algorithmus zu sein.

Auswertung der Phaseninformation
Zudem wäre ein alternativer Indikator für das Verzerrungsmaß, welcher nicht auf der Klirrfaktorberechnung basieren würde, eine interessante Herausforderung für eine weitere Arbeit. Als Basis hierfür wäre es von Interesse zu sehen, ob die Phaseninformation der Spektralanteile genutzt werden kann, um auf den Arbeitspunkt des Sensors zu schließen. Der für diese Arbeit aufgenommene repräsentative Messdatensatz (siehe Abschnitt 5.3) könnte dazu verwendet werden weitere Phasenuntersuchungen an den realen ABS-Sensorsignalen durchzuführen.

Optimierung der RAFT
Bei genauerer Betrachtung der Transformationsmatrizen in Abschnitt 4.3.2 konnte man sehen, dass die Cob- und Sib-Matrizen die Fourier-Koeffizienten liefern. Dabei stellte sich die interessanteste Frage, ob eine Spektralschätzung möglicherweise durchführbar wäre, ohne die Abtastrate auf 240 Samples pro Erfassungzeitfenster zu erhöhen.

Hierzu gilt es zu überprüfen, ob die Anzahl der Samples z.B. auf 64 Abtastwerte pro Erfassungszeitraum gesetzt werden können, indem man nicht die Fourier-Reihenentwicklung als Basis für die Korrektur nutzt.

Zur spektralen Korrektur könnte die Selbstähnlichkeit der Transformationsmatrizen genutzt werden, um neue Koeffizienten für die DFT-Approximation zu berechnen. Sofern dies umsetzbar wäre, ermöglichte es eine noch effizientere Berechnung der DFT-Approximation in einem digitalen System. Die notwendingen Ressourcen könnten damit auf ein Minimum reduziert werden.

Glossar

Sprachliche Abkürzungen

ABS	Antiblockiersystem
AFT	Arithmetische Fourier-Transformation
AMR	Anisotroper Magnetoresistiver Effekt
ASIC	Anwendungsspezifische integrierte Schaltung (Application Specific Integrated Circuit)
Cadence ERC	Cadence Encounter RTL Compiler
CMOS	Complementary Metal Oxide Semiconductor
DFT	Diskrete Fourier-Transformation
ESZ-ABS	Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren
Frequenzverdopplung	Als Frequenzverdopplung wird im Forschungsprojekt ESZ-ABS eine so stark ausgeprägte zweite Harmonische verstanden, die dazu führt, dass die darunterliegende Grundwelle nicht mehr trivial erkannt werden kann
HDI	Harmonic Distortion Infinite: ein Verfahren, welches die Leistung der Grundwelle als Referenz für die gesamte Harmonische Verzerrung nutzt
$\mathbf{HD_K}$	Harmonic Distortion K: ein Verfahren, welches eine begrenzte Anzahl an Harmonischen zur Berechnung eines Verzerrungsmaßes verwendet
HDL	Hardwarebeschreibungssprache (Hardware Description Language)
HIT-KIT	Das HIT-KIT umfasst die von *austriamicrosystems* zur Verfügung gestellten Standardzellenbibliotheken sowie verschiedene Skripte für die Benutzung der Cadence Toolchain

QDFT	Grob quantisierte DFT
RAFT	Rectangular Approximated Fourier-Transform
RTL	Register-Transfer Level
THD	Gesamt Harmonische Verzerrung (Total Harmonic Distortion)
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Formelzeichen

H_k	Die k. Harmonische
N	Anzahl der Samples
K	Anzahl der Harmonischen
n	Zeit- bzw. Winkelindex
k	Frequenz- bzw. Ordnungsindex
x_n	Reelle diskrete Eingangsfolge $0 \leq n < N$
X	DFT-Transformierte von x_n
R	RAFT-Transformierte von x_n
$\mathbf{A}$	Verzerrungsmatrix Realteil
$\mathbf{B}$	Verzerrungsmatrix Imaginärteil
$\mathbf{A}^{-1}$	Korrekturmatrix Realteil
$\mathbf{B}^{-1}$	Korrekturmatrix Imaginärteil

Literaturverzeichnis

[1] BRIGGS, William L. ; HENSON, Van E.: *The DFT: An Owners' Manual for the Discrete Fourier Transform*. Society for Industrial and Applied Mathematics, 1 1987. – ISBN 9780898713428

[2] DRESCHHOFF, Jan-Heiner: *VHDL-Implementierung des Ausgabeprotokolls von ABS-Sensoren*, HAW Hamburg, Studienarbeit, 2009

[3] DRESCHHOFF, Jan-Heiner: *FPGA-Prototyp der Signalverarbeitung für ABS-Sensoren mit Diagnosefunktion als VHDL-Implemenation*, HAW Hamburg, Diploma thesis, 2010

[4] GAUSS, Eugen: *WALSH-Funktionen für Ingenieure und Naturwissenschaftler*. 1994. Vieweg+Teubner Verlag, 6 1994. – ISBN 9783519020998

[5] GOERTZEL, G.: An algorithm for the evaluation of finite trigonometric series. In: *The American Mathematical Monthly* 65 (1958), Nr. 1, S. 34–35

[6] GÖPEL, Wolfgang ; JOACHIM HESSE ; J.N. ZEMEL ; RICHARD BOLL ; KENNETH J. OVERSHOTT ; (Hrsg.): *Magnetic Sensors: A Comprehensive Survey (Sensors a Compre- hensive Survey)*. Bd. 5. Wiley-VCH, 1990

[7] IFEACHOR, Emmanuel ; JERVIS, Barrie: *Digital Signal Processing: A Practical Approach (2nd Edition)*. 2. Prentice Hall, 10 2001. – ISBN 9780201596199

[8] IVANOV, Kalin: *Fehlersichere Automatisierung eines Encoder-Messplatzes zur Untersuchung von ABS-Sensoren*, HAW Hamburg, Diploma thesis, 2011

[9] JEGENHORST, Niels: *Entwicklung eines Controllersystems zur Zustandserkennung von ABS-Sensoren*, HAW Hamburg, Diploma thesis, 2009

[10] KAMMEYER, K.D. ; KROSCHEL, K.: *Digitale Signalverarbeitung*. Vieweg + Teubner, 2009. – ISBN 9783834806109

[11] KAVAIYA, Gaurang: *AN257 - DTMF Detection Using PIC18 Microcontrollers*. 2005. – URL `http://ww1.microchip.com/downloads/en/AppNotes/MCHP_DTMFDetv1.2.zip`. – Microchip

[12] KOCH, Lennart: *Aufwandsminimierte Schätzung von Harmonischen zur Zustandsbestimmung von ABS-Sensoren*, HAW Hamburg, Diploma thesis, 2010

[13] KREY, M. ; RIEMSCHNEIDER, K.-R.: Diagnose von magnetischen Drehzahlsensoren durch fortlaufende Harmonischen-Analyse. In: *Internationales Forum Mechatronik*. Cham, Germany, September 2011, S. 93–105

[14] KREY, M. ; RIEMSCHNEIDER, K.-R. ; ZIPPEL, S.: Signal synthesis for magnetoresistive speed sensors based on field simulations combined with measured sensor characteristic diagrams. In: *Instrumentation and Measurement Technology Conference (I2MTC), 2012 IEEE International*, may 2012, S. 300 –305. – ISSN 1091-5281

[15] KREY, Martin: *Systemarchitektur und Signalverarbeitung für die Diagnose von magnetischen ABS-Sensoren*. Vortrag an der Helmut-Schmidt-Universität (HSU). Juni 2012

[16] KREY, Martin: *Systemarchitektur und Signalverarbeitung für die Diagnose von magnetischen ABS-Sensoren (in Bearbeitung)*, HAW Hamburg / HSU Hamburg, Dissertation, geplant 2013

[17] KREY, Martin ; RIEMSCHNEIDER, Karl-Ragmar: Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren / HAW Hamburg. 2010. – Forschungsbericht

[18] KREY, Martin ; RIEMSCHNEIDER, Karl-Ragmar: Selbst-Diagnose von ABS-Sensoren mittels integrierter Signalverarbeitung / HAW Hamburg. 2010. – Forschungsbericht

[19] MEYER-BÄSE, U.: *Schnelle Digitale Signalverarbeitung: Algorithmen, Architekturen, Anwendungen*. Springer-Verlag GmbH, 2000. – ISBN 9783540676621

[20] PAPULA, L.: *Mathematische Formelsammlung: für Ingenieure und Naturwissenschaftler*. Vieweg+Teubner Verlag, 2009 (Lothar Papula). – ISBN 9783834807571

[21] PHILIPS SEMICONDUCTORS: *General Rotational speed measurement*. 1998

[22] PIOREK, Markus: *Hard- und Software eines Messsystems zur Harmonischenanalyse bei magnetischen Winkelsensoren*, HAW Hamburg, Diploma thesis, 2011

[23] POPPINGA, Heiko: *Controller-Implementation und messtechnische Erprobung der Signalverarbeitung für die Diagnosefunktion von ABS-Sensoren*, HAW Hamburg, Bachelor's thesis, 2011

[24] RIEMSCHNEIDER, Karl-Ragmar: *Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren (ESZ-ABS) - Forschungsantrag*. IngenieurNachwuchs2008. 2008

[25] RIEMSCHNEIDER, Karl-Ragmar ; KREY, Martin: Signalverarbeitung zur Funktionsdiagnose bei magnetischen Sensoren. In: *EForum* (2011), S. 20–25

[26] RIEMSCHNEIDER, Karl-Ragmar ; RETTIG, Rasmus ; KREY, Martin ; SABOTTA, Daniel ; ZAHN, Fabian: *Development of Advanced Diagnostic Functions in Very High Volume Automotive Sensor Applications*. Veröffentlichung in Vorbereitung (IEEE SoCC). 2013

[27] SABOTTA, Daniel: *Design eines CMOS-Testchips für die digitale und analoge Signalverarbeitung in ABS-Sensoren*, HAW Hamburg, Master's thesis, 2013

[28] SCHOERMER, Christian: *AMR-Messbrücken für ABS-Sensoren*, HAW Hamburg / NXP Semiconductors, Studienarbeit, 2008

[29] SCHOERMER, Christian: *Automatisierter Radmessplatz für ABS-Sensoren mit aktiven und passiven Encodern verschiedener Automobil-Hersteller*, HAW Hamburg, Diploma thesis, 2010

[30] SMITH, Julius O.: *Mathematics of the Discrete Fourier Transform (DFT): with Audio Applications —- Second Edition*. 2. W3K Publishing, 4 2007. – ISBN 9780974560748

[31] STAHL, Martin: *Controllersystem zur Verstärkungsregelung und Offsetkompensation für ABS-Sensoren mit Diagnosefunktion*, HAW Hamburg, Bachelor's thesis, 2010

[32] SUNDARARAJAN, D.: *The Discrete Fourier Transform: Theory, Algorithms and Applications*. World Scientific, 2001. – ISBN 9789810245214

[33] TEMME, S.: Audio distortion measurements. In: *Application Note, Bruel & Kjar* (1992)

[34] WERNER, M.: *Digitale Signalverarbeitung Mit MATLAB®: Grundkurs Mit 16 Ausführlichen Versuchen*. Vieweg Verlag, Friedr, & Sohn Verlagsgesellschaft mbH, 2011. – ISBN 9783834886217

[35] ZIPPEL, Stefan: *Simulation des magnetischen Systems bei ABS-Sensoren*, HAW Hamburg, Master's thesis, 2011

Anhang

A. Matlab Quellcodes

A.1. RAFT-Implementierung

```
function res = sib(x,n)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Binary sine function values {-1, 1} which is defined
% in the left-closed interval [0, 1)
%
% sib(x, n) is the n'th binary cosine with n periods in
% the interval [0, 1)
%
% -> feed this function only with values within (0 <= x < 1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% If no order is specified set it to the first
if nargin < 2
    n = 1;
end

% if n is not a scalar throw an error
if any(size(n)~=1),
    error(generatemsgid('SignalErr'),'The parameter ''n'' must be a scalar.')
end

if(n==0)
%     res = -ones(1, length(x));
    res = zeros(1, length(x));
    return
end;

for ii=1:length(x)
    tmp(ii) = (-1)^(floor((2*n)*x(ii)));
end

res = tmp;
```

Listing A.1: sib.m - Binärer Sinus

```
function res = cob(x,n)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Binary cosine function values {-1, 1} which is defined
% in the left-closed interval [0, 1)
%
% cob(x, n) is the n'th binary cosine with n periods in
% the interval [0, 1)
%
% -> feed this function only with values within (0 <= x < 1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% If no order is specified set it to the first
if nargin < 2
    n = 1;
end

% if n is not a scalar throw an error
if any(size(n)~=1),
    error(generatemsgid('SignalErr'),'The parameter ''n'' must be a scalar.')
end

if(n==0)
    res = ones(1, length(x));
    return
end

for ii=1:length(x)
    tmp(ii) = (-1)^(floor( ((2*n)*x(ii)+0.5) ));
end

res = tmp;
```

Listing A.2: cob.m - Binärer Cosinus

```
function res = raft_correction_matrices( h )
% Calculate the correction matrices for the real and the imaginary part of
% the raft depending on the number of harmonics used
%
% input h is the number of harmonics to be calculated
%
% returns the (h+1) by (h+1) matrices in a struct

A = eye(h+1);

A(1,1) = (pi/4);

% c-style matrix adressing
for ii=1:h+1
    sign = -1;
    for jj=1:h+1
        indexjj = (ii*(2*jj+1));
        if  indexjj < h+1
            A(ii+1, indexjj+1) = (sign)/(2*jj+1);
            sign = sign * (-1);
        end
    end
end
% correction factor from fourier series expansion
A = (4/pi) * A;
% b is simply the absolute of a
B = abs(A);

%the final matrices are A^-1 and B^-1
Ainv = A^-1;
Binv = B^-1;

res = struct('A', Ainv, 'B', Binv, 'A_ref', A, 'B_ref', B);
end
```

Listing A.3: raft_correction_matrices.m - Berechnung der Korrekturmatrizen

```
function res = raft2(x,h,n)
% Rectangular Approximated Fourier Transformation (RAFT) for real vectors
%
% x - signal vector
% h - number of harmonics to analyse
%
% DFT on a real vector:
%       X[k] = SUM( x[k] * exp(-j*2*PI*(k/N)*n) )
%
%       X[k] =       SUM( x[k] * cos(2*PI*(k/N)*n) )
%              - j * SUM( x[k] * sin(2*PI*(k/N)*n) )
%
% RAFT on a real vector:
%
%       X[k] =       SUM( x[k] * cob(2*PI*(k/N)*n) )
%              - j * SUM( x[k] * sib(2*PI*(k/N)*n) )
%
% Note: this transform was designed to calculate the harmonic components of
%       an order tracked system feed the function with integer multiples of
%       the period from the sample signal currently 6 harmonics will be
%       calculated with a signal consisting of 120 samples
%
% Update 06.09: added scaling factor for the matrices
%        07.09: replaced dft with fft for speed purposes
%
% res = raft(x,h,f)
persistent sib_table;
persistent cob_table;
% persistent sin_table;
% persistent cos_table;

persistent corr_matrices;

n_harmonics = h;
n_samples   = n;

% angle vector
angle_vec   = 0:(1/n_samples):(1-(1/n_samples));

%% Tables and correction matrices for n_harmonics
if (isempty(sib_table) || isempty(cob_table) || isempty(corr_matrices)) % || isempty(sin_table) || ...
      isempty(cos_table) )

    % calc tables for sir and cor
    sib_table = zeros(n_harmonics+1,n_samples);
    cob_table = zeros(n_harmonics+1,n_samples);

    for ii=0:n_harmonics;
        sib_table(ii+1,:) = sib(angle_vec,ii);
        cob_table(ii+1,:) = cob(angle_vec,ii);
    end

    %get correction matrices for the raft
    corr_matrices = raft_correction_matrices(n_harmonics);
end;

%% RAFT -> n_harmonics+1 ROWS (DC + n_harmonics)
real_part =  1/n_samples * (cob_table * x');
imag_part = -1/n_samples * (sib_table * x');

%% DFT -> n_samples ROWS
dft = 1/n_samples * fft(x);
% real_part_dft =  1/n_samples * (cos_table * x');
% imag_part_dft = -1/n_samples * (sin_table * x');
real_part_dft = real(dft);
imag_part_dft = imag(dft);


%% correction of the real and the imaginary part
real_part = corr_matrices.A * real_part;
imag_part = corr_matrices.B * imag_part;

res = struct(   'real_part', real_part', ...
                'imag_part', imag_part', ...
                'real_part_dft', real_part_dft, ...
                'imag_part_dft', imag_part_dft);
```

Listing A.4: raft2.m - Implementierung des RAFT-Algorithmus

```
function res = raft2_fixed(x,h,n)
% Rectangular Approximated Fourier Transformation (RAFT) for real vectors
% fixed point version
% x - signal vector
% h - number of harmonics to analyse
% n - samples of the signal vector
%
% Note: this transform was designed to calculate the harmonic components of
%       an order tracked system feed the function with integer multiples of
%       the period from the sample signal currently up to 6 harmonics will be
%       calculated with a signal consisting of 240 samples
%
% res = raft2_fixed(x,h,n) the signature is to ensure that you can swap
% pointers to the transfer function from @raft2 -> @raft2_fixed
%
persistent sib_table;
persistent cob_table;
persistent sin_table;
persistent cos_table;

n_bits = 10;
n_harmonics = 5; % <- FIXED!! (6 is also possible)
n_samples   = n;

% angle vector
angle_vec   = 0:(1/n_samples):(1-(1/n_samples));

% fixed point conversion factor
fac = 2^(n_bits-1);

%convert input to 10 bits fixed
x = round(x * (fac-1));

%% Tables and correction matrices for n_harmonics (setup)
if (isempty(sib_table) || isempty(cob_table) || isempty(corr_matrices) || isempty(sin_table) || ...
      isempty(cos_table) )

    % calc tables for sir and cor
    sib_table = zeros(n_harmonics+1,n_samples);
    cob_table = zeros(n_harmonics+1,n_samples);

    for ii=0:n_harmonics;
        sib_table(ii+1,:) = sib(angle_vec,ii);
        cob_table(ii+1,:) = cob(angle_vec,ii);
    end

     % create reference dft matrix
     dft_matrix = fft(eye(n_samples));
     % conversion to fixedpoint
     dft_matrix = round(dft_matrix*(fac-1));

     % split imaginary and real part into cos and sin tables
     sin_table = -imag(dft_matrix);
     cos_table = real(dft_matrix);
end;

% SOT -> start of transformation
%% RAFT -> n_harmonics+1 ROWS (DC + n_harmonics)
real_part = floor( (cob_table * x'));
imag_part = floor( -(sib_table * x'));
%% DFT -> n_samples ROWS
real_part_dft = floor(  (cos_table * x'));
imag_part_dft = floor( -(sin_table * x'));
%% correction of the real and the imaginary part
% correction values -> fp format
one_third = round(1/3*(fac-1));
one_fifth = round(1/5*(fac-1));

real_part(2) = real_part(2) + fix((one_third * real_part(4))/fac) - fix((one_fifth * real_part(6))/fac);
imag_part(2) = imag_part(2) - fix((one_third * imag_part(4))/fac) - fix((one_fifth * imag_part(6))/fac);

% RESULT STRUCT
res = struct(   'real_part', real_part', ...
                'imag_part', imag_part', ...
                'real_part_dft', real_part_dft(1:n_harmonics+1), ...
                'imag_part_dft', imag_part_dft(1:n_harmonics+1), ...
                'real_part_squared', (real_part'.*real_part'), ...
                'imag_part_squared', (imag_part'.*imag_part'));
% EOT -> end of transformation
```

Listing A.5: raft2_fixed.m - Implementierung des RAFT-Algorithmus in vereinfachter Fest-Komma Darstellung

A.2. Berechnung der Verzerrungsmaße

```
function res = calc_thd( x )
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HDI Calculation used in ESZ-ABS project
% x = [H1 H2 H3 ... HN]
% -> no dc in x please
%
% FZ 2012
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

num = sum(x(2:end).^2);
denum = sum(x(1:end).^2);
res = 100 * sqrt( num/denum );
end
```

Listing A.6: calc_thd.m - Berechnung des HD_K Verzerrungsmaßes

```
function res = calc_hdinf( x, h )
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HDI Calculation used in ESZ-ABS project
% x = time signal
% h = harmonic
%
% FZ 2012
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

spec = (1/length(x) * abs(fft(x))).^2;

dc_power = (1/length(x) * sum(x))^2;  %[V^2]
ms_power = 1/length(x) * sum(x.^2);   %[V^2]

h_power = 2*spec(h+1);  %[V^2]

res = 100*sqrt( 1-h_power/(ms_power - dc_power) );

end
```

Listing A.7: calc_hdinf.m - Berechnung des *HDI* Verzerrungsmaßes

A.3. Monte-Carlo-Simulationen

```
% Analyze the error of the amplitudes of the different harmonics in the
% signal
clear all;
close all;
%% EDITABLE PARAMETERS
% harmonics that should be analyzed (choose a integer values < 11)
n_harmonics = 6; % DO NOT CHANGE THIS PLEASE!
% number of cycles to be analyzed
N = 1e6;

%% SETUP
n_samples = 240;
t_vec = 0:1/(n_samples):1-1/(n_samples);

pt = struct('real_part', [], 'imag_part', [], 'real_part_dft', [], 'imag_part_dft', []);
t = repmat(pt,1,N);

p = 0;
fprintf('\n');
q = [];

%% ANALYSIS
tic;
for i=1:N
    % gen testsig
    testsig = zeros(1, n_samples);
    for ii=1:n_harmonics;
        testsig = testsig + (rand()) *2* sin(2*pi*ii*t_vec + rand()*pi);
    end

    % transform
    t(i) = raft2(testsig, n_harmonics, n_samples);

    q(i).signal = testsig;

    abs_raft = abs(complex(t(i).real_part,     t(i).imag_part));
    abs_dft  = abs(complex(t(i).real_part_dft, t(i).imag_part_dft));

    q(i).dft_thd = calc_thd(abs_dft(2:n_harmonics+1));
    q(i).raft_thd = calc_thd(abs_raft(2:end));

    q(i).dft_hdi  = calc_hdinf(q(i).signal, 1);
    q(i).raft_hdi = calc_hdinf(q(i).signal, 1);

    q(i).dft_spec   = abs(complex(t(i).real_part_dft(1:n_harmonics+1), t(i).imag_part_dft(1:n_harmonics+1)));
    q(i).raft_spec  = abs(complex(t(i).real_part(1:n_harmonics+1), t(i).imag_part(1:n_harmonics+1)));
    % sum of squared errors
    q(i).spec_diff = sum((q(i).dft_spec - q(i).raft_spec).^2);

    % output progress
    if mod(i,N/100) == 0
        fprintf('.');
        p = p+1;
    end
end
fprintf('\ncasino took:\n____________________\n');toc;fprintf('\n');

figure;
subplot(2,1,1);
hist([q.dft_thd] - [q.raft_thd], 1000)
title('Deviation (HD_{6,DFT} - HD_{6,RAFT})');
xlabel('HD Deviation [%]');
ylabel('Occurrences');
% center it
ax = axis();
mx = max(abs(ax(1:2)));
xlim([-mx mx]);

subplot(2,1,2);
hist([q.spec_diff], 1000);
title('Spectral Errors');
xlabel('Sum of Squared Errors of the Harmonics');
ylabel('Occurrences');
export_fig(gcf, 'mc_with_limited_harm.pdf', 'A4L');
```

Listing A.8: mc_with_limited_harm.m - Monte-Carlo-Simulation mit eingeschränkten Harmonischen

```
% Analyze the error of the amplitudes of the different harmonics in the
% signal
clear all;
close all;
%% EDITABLE PARAMETERS
% harmonics that should be analyzed (choose a integer values < 11)
n_harmonics = 6; % DO NOT CHANGE THIS PLEASE!
% number of cycles to be analyzed
N = 1e6;

%% SETUP
n_samples = 240;
t_vec = 0:1/(n_samples):1-1/(n_samples);

pt = struct('real_part', [], 'imag_part', [], 'real_part_dft', [], 'imag_part_dft', []);
t = repmat(pt,1,N);

p = 0;
fprintf('\n');
q = [];

%% ANALYSIS
tic;
for i=1:N
    % gen testsig
    testsig = zeros(1, n_samples);
    for ii=1:31;
        testsig = testsig + (rand()) *2* sin(2*pi*ii*t_vec + rand()*pi);
    end

    % transform
    t(i) = raft2(testsig, n_harmonics, n_samples);
    q(i).signal = testsig;

    abs_raft = abs(complex(t(i).real_part,     t(i).imag_part));
    abs_dft  = abs(complex(t(i).real_part_dft, t(i).imag_part_dft));

    q(i).dft_thd = calc_thd(abs_dft(2:n_harmonics+1));
    q(i).raft_thd = calc_thd(abs_raft(2:end));

    q(i).dft_hdi  = calc_hdinf(q(i).signal, 1);
    q(i).raft_hdi = calc_hdinf(q(i).signal, 1);

    q(i).dft_spec   = abs(complex(t(i).real_part_dft(1:n_harmonics+1), t(i).imag_part_dft(1:n_harmonics+1)));
    q(i).raft_spec  = abs(complex(t(i).real_part(1:n_harmonics+1), t(i).imag_part(1:n_harmonics+1)));
    % sum of squared errors
    q(i).spec_diff = sum((q(i).dft_spec - q(i).raft_spec).^2);

    % output progress
    if mod(i,N/100) == 0
        fprintf('.');
        p = p+1;
    end
end
fprintf('\ncasino took:\n____________________\n');
toc;
fprintf('\n');

figure;
subplot(2,1,1);
hist([q.dft_thd] - [q.raft_thd], 1000)
title('Deviation (HD_{6,DFT} - HD_{6,RAFT})');
xlabel('HD Deviation [%]');
ylabel('Occurrences');
% center it
ax = axis();
mx = max(abs(ax(1:2)));
xlim([-mx mx]);

subplot(2,1,2);
hist([q.spec_diff], 1000);
title('Spectral Errors');
xlabel('Sum of Squared Errors of the Harmonics');
ylabel('Occurrences');
export_fig(gcf, 'mc_with_more_harm.pdf', 'A4L');
```

Listing A.9: mc_with_more_harm.m - Monte-Carlo-Simulation mit einem größeren Umfang an Harmonischen

```
% Analyze the error of the amplitudes of the different harmonics in the
% signal
clear all;
close all;
%% EDITABLE PARAMETERS
% harmonics that should be analyzed (choose a integer values < 11)
n_harmonics = 4; % DO NOT CHANGE THIS PLEASE!
% number of cycles to be analyzed
N = 1e6;
%% SETUP
n_samples = 240;
t_vec = 0:1/(n_samples):1-1/(n_samples);

pt = struct('real_part', [], 'imag_part', [], 'real_part_dft', [], 'imag_part_dft', []);
t = repmat(pt,1,N);

p = 0;
fprintf('\n');
q = [];

%% ANALYSIS
tic;
max_measurements = [1.00000000000000 1.08627875079422 0.732896147001951 0.310069768697789 0.205017977795444 ...
    0.162976398723030 0.105308187014433 0.0808428323255261 0.0650982570905788 0.0494988802635683];
for i=1:N
    % gen testsig
    %testsig = zeros(1, length(t_vec));
    testsig = sin(2*pi*t_vec);
    for ii=2:length(max_measurements)
        testsig = testsig + (max_measurements(ii)*rand()) *2* sin(2*pi*ii*t_vec + rand()*pi);
    end

    % transform
    t(i) = raft2(testsig, n_harmonics, n_samples);
    q(i).signal = testsig;

    abs_raft = abs(complex(t(i).real_part,     t(i).imag_part));
    abs_dft  = abs(complex(t(i).real_part_dft, t(i).imag_part_dft));

    q(i).dft_thd = calc_thd(abs_dft(2:n_harmonics+1));
    q(i).raft_thd = calc_thd(abs_raft(2:end));

    q(i).dft_hdi  = calc_hdinf(q(i).signal, 1);
    q(i).raft_hdi = calc_hdinf(q(i).signal, 1);

    q(i).dft_spec   = abs(complex(t(i).real_part_dft(1:n_harmonics+1), t(i).imag_part_dft(1:n_harmonics+1)));
    q(i).raft_spec  = abs(complex(t(i).real_part(1:n_harmonics+1), t(i).imag_part(1:n_harmonics+1)));
    % sum of squared errors
    q(i).spec_diff = sum((q(i).dft_spec - q(i).raft_spec).^2);

    % output progress
    if mod(i,N/100) == 0
        fprintf('.');
        p = p+1;
    end
end
fprintf('\ncasino took:\n____________________\n');
toc;
fprintf('\n');

figure;
subplot(2,1,1);
hist([q.dft_thd] - [q.raft_thd], 1000)
title('Deviation (HD_{6,DFT} - HD_{6,RAFT})');
xlabel('HD Deviation [%]');
ylabel('Occurrences');
% center it
ax = axis();
mx = max(abs(ax(1:2)));
xlim([-mx mx]);

subplot(2,1,2);
hist([q.spec_diff], 1000);
title('Spectral Errors');
xlabel('Sum of Squared Errors of the Harmonics');
ylabel('Occurrences');
export_fig(gcf, 'mc_with_envelope.pdf', 'A4L');
```

Listing A.10: mc_with_envelope.m - Monte-Carlo-Simulation mit Hüllkurvengewichtung der Harmonischen

A.4. VHDL-Codegenerierung

```
clear all;
close all;
%% setup
% samples
N = 240;
% time vector
t_vec = 0:1/N:1-1/N;
% scale factor
scale_factor = (2^9)-1;


%% savenames
sine_op = 'sine_op.stimuli';
sine_tp = 'sine_tp.stimuli';

rect_op = 'rect_op.stimuli';
rect_tp = 'rect_tp.stimuli';

square_op = 'square_op.stimuli';
square_tp = 'square_tp.stimuli';

%% generate stimulus signals
sig_sine_op = round(scale_factor * sin(2*pi*1*t_vec))';
sig_sine_tp = round(scale_factor * sin(2*pi*2*t_vec))';

sig_rect_op = [bartlett(N/2)' -bartlett(N/2)'];
sig_rect_op = round(scale_factor * sig_rect_op)';
sig_rect_tp = [bartlett(N/4)' -bartlett(N/4)' bartlett(N/4)' -bartlett(N/4)'];
sig_rect_tp = round(scale_factor * sig_rect_tp)';


sig_square_op = round(scale_factor * sib(t_vec, 1))';
sig_square_tp = round(scale_factor * sib(t_vec, 2))';


%% write output files
dlmwrite(sine_op, sig_sine_op);
dlmwrite(sine_tp, sig_sine_tp);

dlmwrite(rect_op, sig_rect_op);
dlmwrite(rect_tp, sig_rect_tp);

dlmwrite(square_op, sig_square_op);
dlmwrite(square_tp, sig_square_tp);

subplot(3,2,1);
plot(sig_sine_op);
xlim([0 N-1]);ylim([-(scale_factor+50) scale_factor+50]);
subplot(3,2,2);
plot(sig_sine_tp);
xlim([0 N-1]);ylim([-(scale_factor+50) scale_factor+50]);

subplot(3,2,3);
plot(sig_rect_op);
xlim([0 N-1]);ylim([-(scale_factor+50) scale_factor+50]);
subplot(3,2,4);
plot(sig_rect_tp);
xlim([0 N-1]);ylim([-(scale_factor+50) scale_factor+50]);


subplot(3,2,5);
plot(sig_square_op);
xlim([0 N-1]);ylim([-(scale_factor+150) scale_factor+150]);
subplot(3,2,6);
plot(sig_square_tp);
xlim([0 N-1]);ylim([-(scale_factor+150) scale_factor+150]);
```

Listing A.11: raft_toplevel_stimuli.m - Generation von Testbench Stimuli für die RAFT-Implementierung

```
clear all;
n_harmonics = 5;
n_samples = 240;
% angle vector
angle_vec   = 0:(1/n_samples):(1-(1/n_samples));

%% Tables and correction matrices for n_harmonics

% calc tables for sir and cor
sib_table = zeros(n_harmonics,n_samples);
cob_table = zeros(n_harmonics,n_samples);

for ii=1:n_harmonics;
    sib_table(ii,:) = .5*(1+(-1)*sib(angle_vec,ii));
    cob_table(ii,:) = .5*(1+cob(angle_vec,ii));
end




fid = fopen('out.txt', 'w');

%% ARAY IMPLEMENTATION
fprintf(fid, '\n-- generated arrays');
fprintf(fid, '\ntype array_t is array ( 0 to %d ) of std_logic_vector(%d downto 0);', (256)-1, n_harmonics-1);
fprintf(fid, '\nconstant SIB_TAB : array_t := (');

for ii=1:n_samples;
    x = (sib_table(:, ii)');
    string = '"';
    for jj=1:length(x);
        string = [string num2str(x(jj))];
    end
    string = [string '"'];
    string = fliplr(string); % from 0 to 4 -> 4 downto 0
    fprintf(fid, '%d => %s,', ii-1, string);
end
for ii=n_samples+1:256;
    fprintf(fid, '%d => "-----"', ii-1);
    if ((ii ~= 256))
        fprintf(fid, ', ');
    end;
end
fprintf(fid, ');');


fprintf(fid, '\nconstant COB_TAB : array_t := (');

for ii=1:n_samples;
    x = (cob_table(:, ii)');
    string = '"';
    for jj=1:length(x);
        string = [string num2str(x(jj))];
    end
    string = [string '"'];
    string = fliplr(string); % from 0 to 4 -> 4 downto 0
    fprintf(fid, '%d => %s,', ii-1, string);
end
for ii=n_samples+1:256;
    fprintf(fid, '%d => "-----"', ii-1);
    if ((ii ~= 256))
        fprintf(fid, ', ');
    end;
end
fprintf(fid, ');');

%% CASE STATEMENT DECODER

fprintf(fid, '\n-- generated case');
fprintf(fid, '\n\n');
fprintf(fid, '\nsignal sib_out : std_logic_vector(%i downto 0);', n_harmonics-1);
fprintf(fid, '\nsignal cob_out : std_logic_vector(%i downto 0);', n_harmonics-1);


fprintf(fid, '\nsib_mux : process(i_addr)\nbegin\n\tcase i_addr is');
%         WHEN "00"  =>  output <= in0;
for ii=1:n_samples;
    x = (sib_table(:, ii)');
    string = '"';
    for jj=1:length(x);
        string = [string num2str(x(jj))];
    end
    string = [string '"'];
```

```
        string = fliplr(string); % from 0 to 4 -> 4 downto 0
        fprintf(fid, '\n\t\twhen "%s" => sib_out <= %s;', dec2bin((ii-1), 8), string);
88  end
    fprintf(fid, '\n\t\twhen others => sib_out <= "-----"; -- don''t care');
    fprintf(fid, '\n\tend case;\nend process;');

    fprintf(fid, '\n\n');
93
    fprintf(fid, '\ncob_mux : process(i_addr)\nbegin\n\tcase i_addr is');
    %         WHEN "00"  =>  output <= in0;
    for ii=1:n_samples;
        x = (cob_table(:, ii)');
98      string = '"';
        for jj=1:length(x);
            string = [string num2str(x(jj))];
        end
        string = [string '"'];
103     string = fliplr(string); % from 0 to 4 -> 4 downto 0
        fprintf(fid, '\n\t\twhen "%s" => cob_out <= %s;', dec2bin((ii-1), 8), string);
    end
    fprintf(fid, '\n\t\twhen others => cob_out <= "-----"; -- don''t care');
    fprintf(fid, '\n\tend case;\nend process;');
```

Listing A.12: tables_to_vhdl.m - Umsetzung der Sib/Cob Multiplexersignale nach VHDL

A.5. Auswertung der Scope-Daten des RMP3

```
close all;
clear all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This script converts rmp3 measurement data (from scope) into two parts:
%   1. the 2-period time signal
%   2. the first 10 harmonic components in the spectrum
%
% With this data you can synthesize your own signals from the spectrum and
% apply digital signal processing or any other algorithms to the data
%
%
% THIS SCRIPT RELIES HEAVILY ON:
%       -> function rmp3_analyse_scope_data_v2(dir_name) (M. Krey 2012)
%
% HISTORY:
%   01.10.12
%       -> initial commit
%       -> script reads data from rmp3 measurements and extracts it to
%       scope_analysed
%       -> various speed improvements (compared to the old script) and
%       cleanup of the code
%   03.10.12
%       -> measured signal periods will no longer be saved! (they're too
%       large)
%       -> improved the script so it can extract single periods or multiple
%       in the time domain
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% SETUP
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dir_name = '2012_10_04_rmp_3_02_phi_x_phi_y_phi_z_z'
save_name = 'DATA_TEST'

HARM = 6;
% the number of harmonics which shall be stored of the fft should be
% greater than 2*HARM
N_SPEC = 16;

START_FILE = 1;
END_FILE   = 89;

% set this value only to 1 or 2
PERIODS = 2;

% this parameter makes the files hughe
enable_periods_mat = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% DO NOT EDIT ANYTHING BELOW THIS LINE
if ~exist('dir_name','var')
    dir_name = uigetdir(pwd,'Choose directory with data-files');
end

data_dir = textscan(dir_name, '%s', 'delimiter', filesep);
filename = [data_dir{1}{end} '_scope_analysed_v2.rmp_3.mat' ];

list_of_files = what(dir_name);
files = regexp(list_of_files.mat, '.*_part[0-9]+.rmp_3.mat', 'match');
files = [files{:}];
files = sort(files);

% if( exist(fullfile(list_of_files.path, filename), 'file')~=0 )
%     disp([filename ' already exists, skipping...']);
%     return;
% end

if ~isempty( regexp(basename(dir_name), 'logamp', 'once') )
    ext_gain_offset = false;
else
    ext_gain_offset = true;
end

%% LOAD PARAMETERS
load( fullfile(list_of_files.path, files{1}), 'parameters' );

% tooth-frequency of measurement
tooth_frequency= parameters.tooth_frequency;

% Samplerate
fs = parameters.samplerate;
```

```
80
    %% ---- Einlesen der Messdaten

    scope_analysed_buff = [];

85  %% FILE WALKER
    for i=START_FILE:END_FILE

        disp(['loading ' files{i}]);
        load( fullfile(list_of_files.path, files{i}), 'scope', 'demo', 'parameters');
90
        %% SCOPE WALKER
        for j=1:length(scope)

            %% CALC GAIN -> NOT NEEDED IMHO
95          % fuer die Ermittlung der aktuellen Verstaerkung
            if ext_gain_offset
                gain = double(demo(j).gain*25);
            else
                gain = (2^double(demo(j).raw.g_sig.gain))*25;
100         end

            %%
            %u_diff = scope(j).u_hb1-scope(j).u_hb2;
            % hotfix
105         u_diff = scope(j).u_diff;
            u_diff = u_diff - (sum(u_diff)/length(u_diff));


            expected_crossings = round( (length(u_diff) / parameters.samplerate)* ...
110             parameters.tooth_frequency );

            %cro=crossing(u_diff);
            %b=a(2:end)-a(1:end-1);
            [cro_gap cro_raising cro_falling] = get_zero_crossings(u_diff,'gap', 1000);
115
            if ( ...
                    ( length(cro_raising) > (expected_crossings*1.1) ) ...
                    || ...
                    ( length(cro_raising) < (expected_crossings*0.9) ) ...
120             )
                disp(['unexpected frequency @ ' num2str(j)])

                cro_raising = cro_raising(1:2/PERIODS:end);
            end
125


            %% PROCESS DATA -> periods, SPECTRUM
            n_periods = length(cro_raising) - PERIODS;
            longest_period = max(cro_raising(2:end) - cro_raising(1:end-1));
130
            periods_mat = NaN(longest_period, n_periods);
            periods_mat_spec = zeros(N_SPEC, n_periods);

            for k=1:n_periods
135             tmp = u_diff(cro_raising(k):cro_raising(k+PERIODS));
                tmp_spec = fft(tmp)/length(tmp)*2;

                if (enable_periods_mat)
                    periods_mat(1:length(tmp), k) = tmp;
140             end

                periods_mat_spec(:, k) = tmp_spec(2:N_SPEC+1);
            end

145         scope_analysed.x = scope(j).x;
            scope_analysed.y = scope(j).y;
            scope_analysed.z = scope(j).z;
            scope_analysed.phi_x = scope(j).phi_x;
            scope_analysed.phi_y = scope(j).phi_y;
150         scope_analysed.phi_z = scope(j).phi_z;

            scope_analysed.gain = gain;

            % SAVE TIME AND SPECTRUM OF THE ANALYSED SIGNAL
155         if (enable_periods_mat)
                scope_analysed.periods_mat = periods_mat;
            end
            scope_analysed.periods_mat_spec = periods_mat_spec;

160
            if isempty(scope_analysed_buff)
                scope_analysed_buff = scope_analysed;
            else
                scope_analysed_buff = [scope_analysed_buff scope_analysed];
```

```
165             end

            end

        end
170
        scope_analysed = scope_analysed_buff;
        clear scope_analysed_buff;

        disp(['saving ' save_name]);
175     save(save_name, 'scope_analysed', '-v7.3');
        %     save( fullfile(list_of_files.path, filename), 'scope_analysed');
```

Listing A.13: scope_rmp3measurement2data.m - Präprozessor für die Verarbeitung der RMP3-Messdaten

```
clear all;
close all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rmp3measurement_analyze : is a script to analyze the data which was
%                           extracted by rmp3measurement2data
%                           the data is synthesized from the measurement
%                           data using a limited spectrum of the original
%                           signal and reconstructing it in the time domain
%                           with a perfect saturation degree
%
% EXAMPLE SETUP:    n_samples = 120;
%                   n_harmonics = 6;
%                   n_periods = 2;
%                   dev_hd = 3;
%
%
% HISTORY:
%   01.10.12    initial commit
%       -> analysis of the data in thd absolut and relative
%   02.10.12    bugfix release
%       -> measure power of h_even and h_odd -> output f-doubling found
%
% TODO:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'DEFINES'
LINE='\n-------------------------------------------------------------';

% start time measurement
tic;
fprintf('\nrmp3measurement_analyze - F. Zahn 2012');
fprintf(LINE);

%% LOAD DATA
fprintf('\nloading data... ');

load('DATA_TEST.mat')

fprintf('done!');
fprintf('\nprocessing ');

%% SETUP
n_samples = 240;
n_harmonics = 6;
% how many periods are in the measured signal? 1 or 2
n_periods = 2;
% output a warning if greater this dev (in %)
dev_hd = 1;

fdoubling_threshold = 0.1;

% (@raft2, @raft2_fixed)
transform = @raft2;
%% feature enables
% -> frequency doubling
enable_fdoubling = 0;

% -> deviation error messages
enable_deverror = 1;

% -> do not include wrong signals in the measurement
enable_autoclean = 1;
autoclean_threshold = 50; % in [%]

% -> PDF output of the plots
enable_pdfexport = 1;

debug = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% SYSTEM DO NOT TOUCH ANYTHING BELOW THIS LINE %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
t_vec = 0:1/n_samples:1-1/n_samples;

signals = [];
diff_hd_abs = [];
diff_hd_rel = [];
thd_used = [];
dev_array_id = [];
dev_array_val = [];
fdoubling_array_id = [];
fdoubling_array_val = [];

signals_rejected  = 0;
signals_wdeviation = 0;
signals_wfdoubling = 0;
```

```
    index = 1;
    %% ANALYSIS
89  mod = int32(length(scope_analysed)/55);
    for ii=1:length(scope_analysed)
        % progress bar
        if (rem(ii, mod) == 0)
            fprintf('.');
94      end
        for jj=1:length(scope_analysed(ii).periods_mat_spec)
            %% calculate the time signal
            % calculate normalisation factor
            norm_factor = max(abs(scope_analysed(ii).periods_mat_spec(1:end,jj)));
99
            time_sig = 0;
            for kk=1:length(scope_analysed(ii).periods_mat_spec(1:end,jj))
                cplx = scope_analysed(ii).periods_mat_spec(kk, jj);
                amp = (1/norm_factor) * abs(cplx);%amplitude
104             phs = angle(cplx);                %phase
                time_sig = time_sig + amp * cos((2*pi*kk)*t_vec + phs);
            end

            %% information of the signal
109         signals(index).samples = n_samples;
            signals(index).time_sig = time_sig;
            signals(index).id = index;
            signals(index).x = scope_analysed(ii).x;
            signals(index).y = scope_analysed(ii).y;
114         signals(index).z = scope_analysed(ii).z;
            signals(index).phi_x = scope_analysed(ii).phi_x;
            signals(index).phi_y = scope_analysed(ii).phi_y;
            signals(index).phi_z = scope_analysed(ii).phi_z;
            % calculate both raft and dft spec
119         signals(index).raft = transform(time_sig, n_harmonics, n_samples);


            % calc abs-spectrum
            raft_spec = (abs(complex(signals(index).raft.real_part, signals(index).raft.imag_part)));
124         dft_spec  = (abs(complex(signals(index).raft.real_part_dft, signals(index).raft.imag_part_dft)));
            signals(index).raft_spec = raft_spec;
            signals(index).dft_spec = dft_spec;

            % calc hd
129         if (n_periods == 1)
                signals(index).raft_hd = calc_thd(raft_spec(2:1:n_harmonics+1));
                signals(index).dft_hd  = calc_thd(dft_spec(2:1:n_harmonics+1));
            elseif (n_periods == 2)
                signals(index).raft_hd = calc_thd(raft_spec(3:2:n_harmonics+1));
134             signals(index).dft_hd  = calc_thd(dft_spec(3:2:n_harmonics+1));
            end

            % calc hd deviation (abs. and rel.)
            signals(index).diff_hd_abs = signals(index).dft_hd - signals(index).raft_hd;
139         signals(index).diff_hd_rel = signals(index).diff_hd_abs/signals(index).dft_hd;

            %% CUSTOMIZABLE OPTIONS
            % thd deviation error output
            if (enable_deverror)
144             if (abs(signals(index).diff_hd_abs) > dev_hd)
                    if (debug)
                        fprintf('\nabs-deviation found @%d, %f is greater than %f',(signals(index).id), ...
                            abs(signals(index).diff_hd_rel * 100), dev_hd);
                        fprintf('\nrelative error: @%f related to THD_{DFT} = %f, THD_{RAFT} = %f', ...
                            signals(index).diff_hd_rel * 100, signals(index).dft_hd, signals(index).raft_hd);
                        fprintf('\n');
149
                    end

                    signals_wdeviation = signals_wdeviation + 1;
                    dev_array_id  = [dev_array_id signals(index).id];
154                 dev_array_val = [dev_array_val signals(index).diff_hd_abs];
                end
            end

            % detection of frequency doubling due to the non-linear nature of
159         % the system
            if (enable_fdoubling && n_periods == 2)

                even_power = signals(index).dft_spec(3);
                odd_power = signals(index).dft_spec(2);
164             if ((odd_power/even_power)*100 >= fdoubling_threshold)
                    if(debug)
                        fprintf('\npossible f-doubling found @%d', index);
                    end
                    signals(index).fdoubling = 1;
```

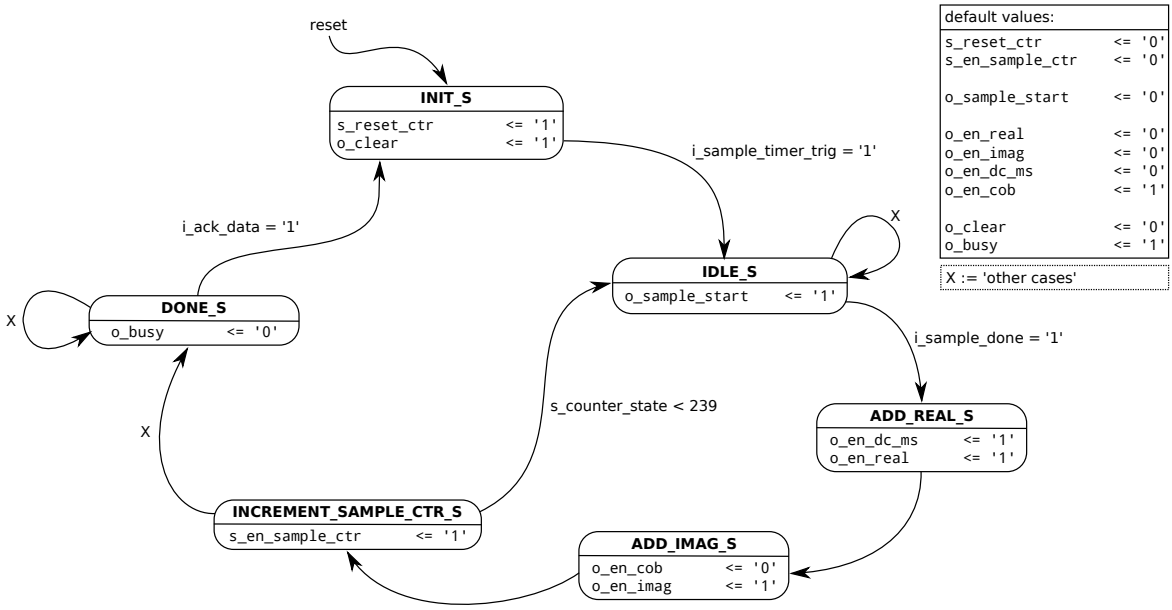
```
169
                    signals_wfdoubling = signals_wfdoubling + 1;
                    fdoubling_array_id  = [fdoubling_array_id signals(index).id];
                    fdoubling_array_val = [fdoubling_array_val signals(index).hdi];
                else
174                 signals(index).fdoubling = 0;
                end
            end

            % autoclean of the measurement base
179         if (enable_autoclean)
                % the number of periods decide where we apply our criterion in the spectrum
                if (n_periods == 1)
                    % the first harmonic must be present and greater than a
                    % specific threshold (keep in mind that an amplitude of 0.5 is 100% of the highest spectral
184                 % component)
                    if ( signals(index).dft_spec(2) >= (.5 * autoclean_threshold/100) )
                        index = index + 1;
                    else
                        signals_rejected = signals_rejected + 1;
189                 end
                elseif (n_periods == 2)
                    % the second harmonic must be present and ... see above
                    if ( signals(index).dft_spec(3) >= (.5 * autoclean_threshold/100) )
                        index = index + 1;
194                 else
                        signals_rejected = signals_rejected + 1;
                    end
                end
            else %autoclean = off
199             index = index + 1; % increment index whether the signal is usable or not
            end
            %display(index)

        end
204 end
    fprintf(' done!');

    %% script breakdown console output
    fprintf('\n\nscript finished in %.2fs and processed %d periods of the signal', toc, (index-1));
209 fprintf(LINE);

    if (enable_autoclean)
        fprintf('\n[autoclean] %d rejected periods of the measured signal', signals_rejected);
    end
214 if (enable_deverror)
        fprintf('\n[deverror]  %d periods with thd deviation found, deviation > %d%%', signals_wdeviation, dev_hd);
    end
    if (enable_fdoubling)
        fprintf('\n[fdoubling] %d periods with possible frequency doubling were found', signals_wfdoubling);
219 end
    fprintf([LINE '\n\n']);


    % TODO -> MOVE THIS TO A POSTPROCESSOR
224 %% GENERATE MEASUREMENT DATA FROM SIGNALS
    bt_index = 1;
    for index=1:length(signals)
        % generate arrays with difference errors
        diff_hd_abs = [diff_hd_abs signals(index).diff_hd_abs];
229     diff_hd_rel = [diff_hd_rel signals(index).diff_hd_rel*100];
        thd_used = [thd_used signals(index).dft_hd];

        % distance -> bathtub plot
    end
234
    %% OUTPUT
    % abs hist error
    close all;

239 % relative hist error
    figure;

    hist(diff_hd_rel, 1000)
    title(['Relative Difference (HD_{DFT} - HD_{RAFT})/(HD_{DFT}) N = ' num2str(n_samples) ' samples']);
244 xlabel('Deviation [%]');
    ylabel('Occurrences');
    if (enable_pdfexport)
        ex_name = [datestr(now, 'yy.mm.dd_HH.MM_') 'rel_difference_hddft-hdraft_n_' num2str(n_samples) '_samples.pdf'];
        export_fig(gcf, ex_name, 'A4L');
249 end

    figure;
    % abs hist error
    subplot(2,1,1);
```

```
254 hist(diff_hd_abs, 1000)
    title(['Absolute Difference (HD_{DFT} - HD_{RAFT}) N = ' num2str(n_samples) ' samples']);
    xlabel('Deviation');
    ylabel('Occurrences');
    % if (enable_pdfexport)
259 %     ex_name = [datestr(now, 'yy.mm.dd_HH.MM_') 'abs_difference_hddft-hdraft_n_' num2str(n_samples) ...
        '_samples.pdf'];
    %     export_fig(gcf, ex_name, 'A5L');
    % end

    % THD hist
264 subplot(2,1,2);
    hist(thd_used, 1000)
    title(['Measured THD (DFT) N = ' num2str(n_samples) ' samples']);
    xlabel('HD_6 [%]');
    ylabel('Occurrences');
269
    if (enable_pdfexport)
        ex_name = [datestr(now, 'yy.mm.dd_HH.MM_') 'abs_diff_and_hdk_hist' num2str(n_samples) '_samples.pdf'];
        export_fig(gcf, ex_name, 'A4L');
    end
274
    %%
    % relative hist error
    figure;
    hist(diff_hd_rel, 50)
279 title('relative difference (HD_{DFT} - HD_{RAFT})/(HD_{DFT})');

    %%
    % dft hd vs raft hd
    figure;hold on;
284 % plot([signals.dft_hd], [signals.raft_hd], '+')
    % plot(0:100,0:100, 'red')
    % xlabel('DFT: HD_6 [%]');
    % ylabel('RAFT: HD_6 [%]');
    % title('DFT vs. RAFT HD_6 from Measured Data')
289 % legend('Measured Values', 'Ideal Characteristic Curve', 'Location','SouthEast');
    % grid on;

    figure;hold on;
    subplot(2,1,1);
294 plot([signals.dft_hd], [signals.raft_hd], '+');hold on;
    plot(0:70,0:70, 'red')
    xlabel('DFT: HD_6 [%]');
    ylabel('RAFT: HD_6 [%]');
    title('DFT vs. RAFT HD_6 from Synthesized Data')
299 legend('Simulated Values', 'Ideal Characteristic Curve', 'Location','SouthEast');
    grid on;

    subplot(2,1,2);
    sorted_dft = sort([signals.dft_hd]);
304 sorted_raft = sort([signals.raft_hd]);
    plot((sorted_dft), sorted_dft-sorted_raft, '+');
    ylabel('Error Difference: DFT - RAFT Indicator [%]');
    xlabel('HD_6 Values from DFT');
    grid on;
309
    %export_fig(gcf, 'zippel_dft_vs_raft_hd6.pdf', 'A4L')
    if (enable_pdfexport)
        ex_name = [datestr(now, 'yy.mm.dd_HH.MM_') 'measured_data_dft_vs_raft_hd6' num2str(n_samples) '_samples.pdf'];
        export_fig(gcf, ex_name, 'A4L');
314 end
```

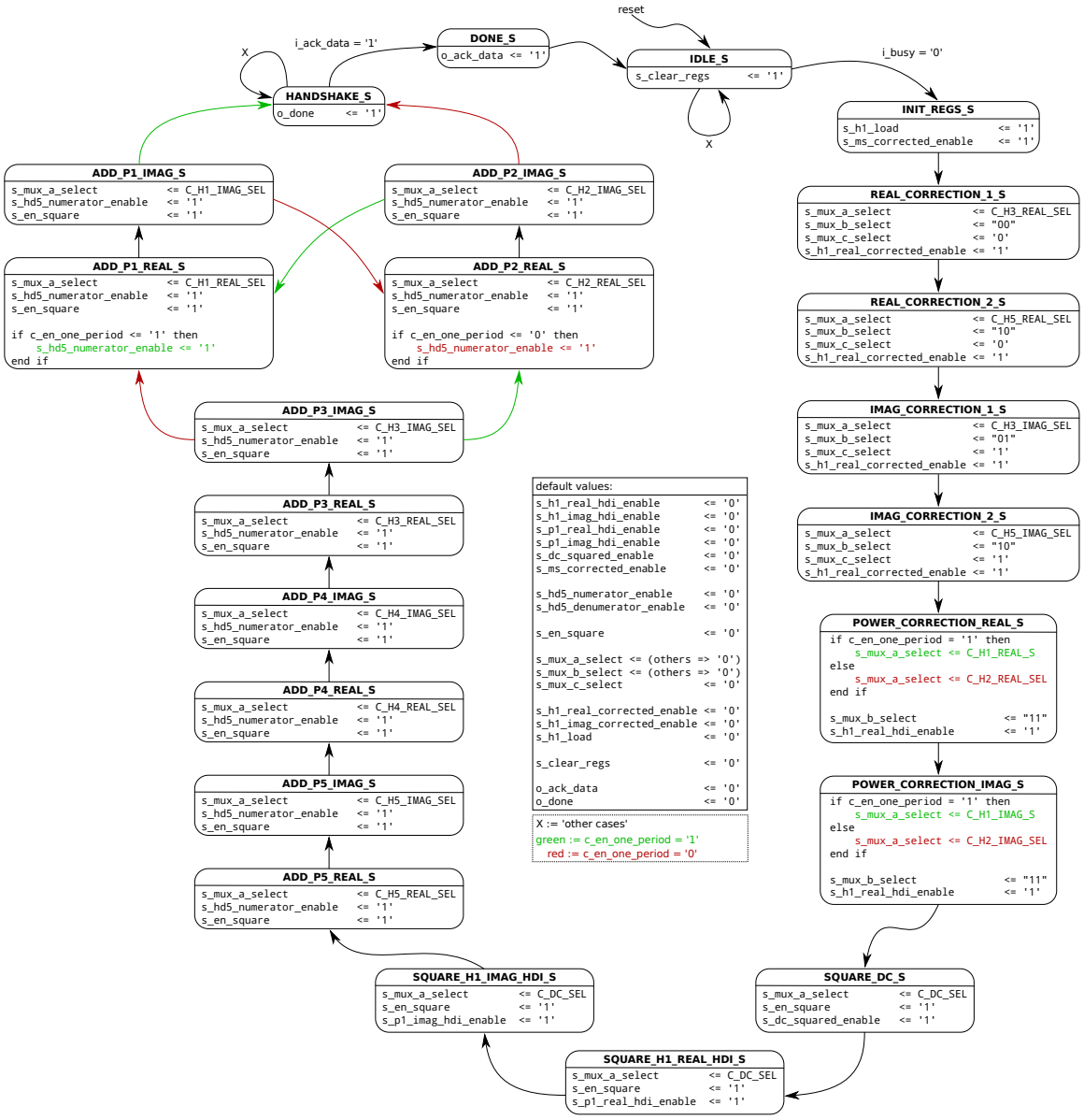
Listing A.14: scope_rmp3measurement_analyze.m - Postprozessor für die Analyse der Daten

B. Dokumentation der digitalen Hardware

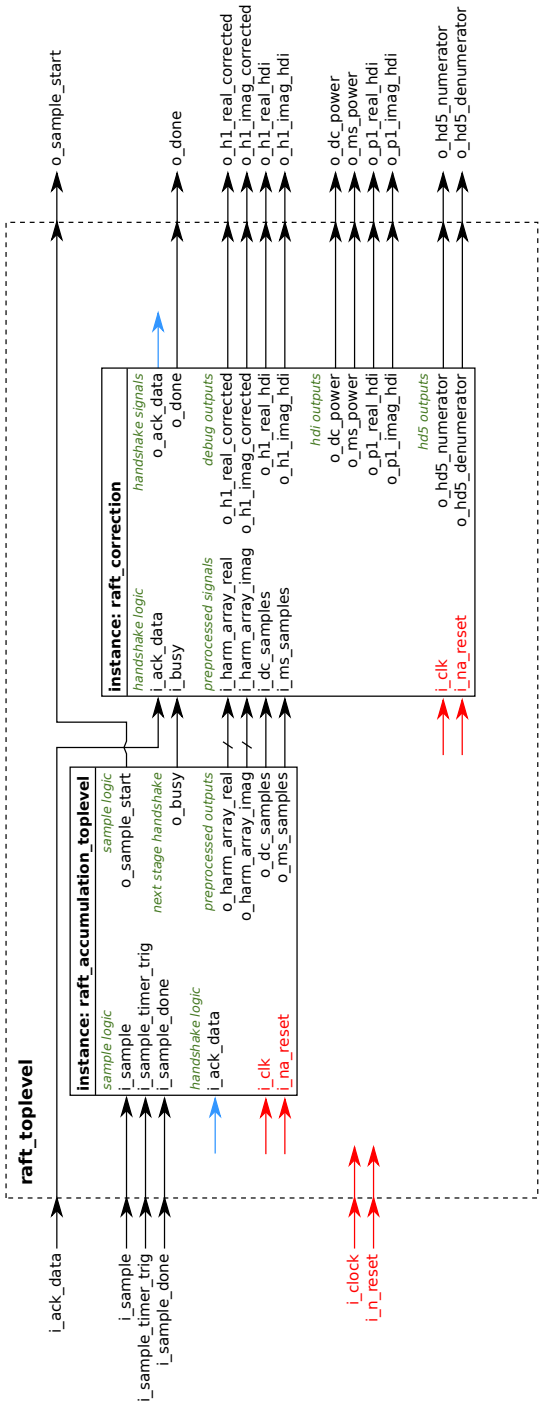
B.1. Zustandsautomat RAFT-Akkumulation



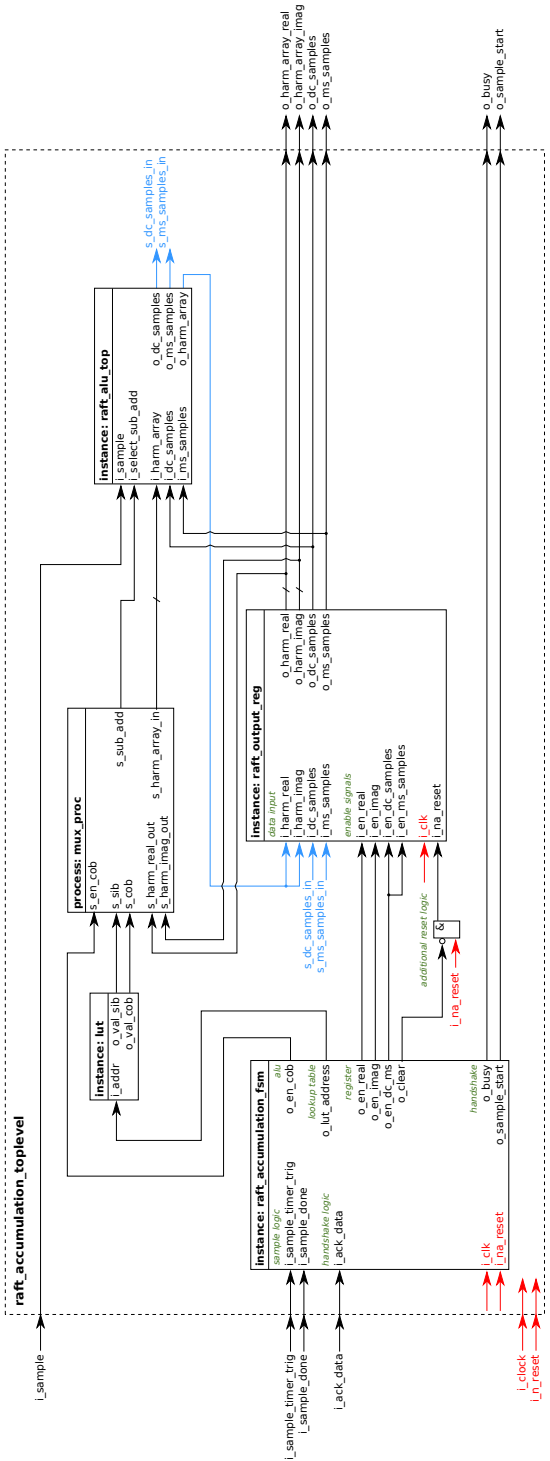
B.2. Zustandsautomat RAFT-Korrektur



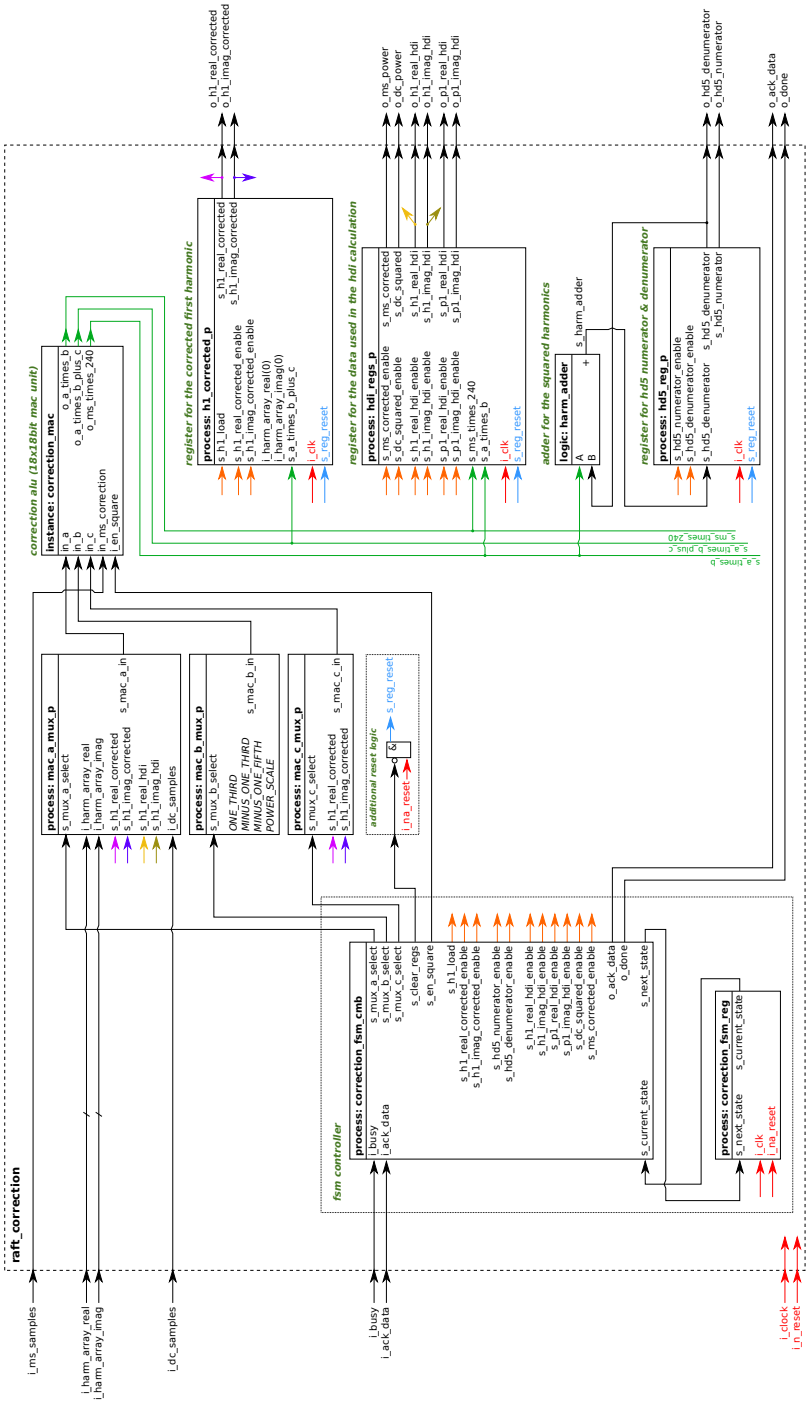
B.3. Blockdiagramm RAFT-Toplevel



B.4. Blockdiagramm RAFT-Akkumulation



B.5. Blockdiagramm RAFT-Korrektur



C. VHDL-Implementierung des RAFT-Verfahrens

C.1. Toplevel

```
1  ----------------------------------------------------------------------------
   -- Entity: raft_toplevel
   ----------------------------------------------------------------------------
   -- Copyright 2012
   -- Filename            : raft_toplevel.vhd
6  -- Creation date       : 2012-12-20
   -- Author(s)           : Fabian Zahn
   -- Version             : 1.00
   -- Description         : Implementation of the overall toplevel
   ----------------------------------------------------------------------------
11 -- File History:
   -- Date          Version   Author    Comment
   ----------------------------------------------------------------------------
   library ieee;
   use ieee.std_logic_1164.ALL;
16 use ieee.numeric_std.ALL;
   use work.raft_pkg.all;


   entity raft_toplevel is
21     port(
           i_clk                : in std_logic;
           i_na_reset           : in std_logic;
           i_sample             : in signed(9 downto 0);
           i_sample_timer_trig  : in std_logic;
26         i_sample_done        : in std_logic;
           i_ack_data           : in std_logic;
           o_done               : out std_logic;
           o_sample_start       : out std_logic;
           -- raft correction debug
31         o_h1_real_corrected  : out signed(17 downto 0);
           o_h1_imag_corrected  : out signed(17 downto 0);
           -- hdi debug
           o_h1_real_hdi : out signed(17 downto 0);
           o_h1_imag_hdi : out signed(17 downto 0);
36         -- hdi
           o_ms_power           : out unsigned(35 downto 0);
           o_dc_power           : out unsigned(35 downto 0);
           o_p1_real_hdi        : out unsigned(35 downto 0);
           o_p1_imag_hdi        : out unsigned(35 downto 0);
41         -- hd5
           o_hd5_numerator      : out unsigned(39 downto 0);
           o_hd5_denumerator    : out unsigned(39 downto 0)
       );
   end entity;
46

   architecture Behavioral of raft_toplevel is

       ------------------------------------------------------------------------
51     -- COMPONENT DECLARATIONS
       ------------------------------------------------------------------------
       -- pre processing
       component raft_accumulation_toplevel
       port(
56         i_clk : in std_logic;
           i_na_reset : in std_logic;
```

```
            i_sample : in signed(9 downto 0);
            i_sample_timer_trig : in std_logic;
            i_sample_done : in std_logic;
            i_ack_data : in std_logic;
            o_sample_start : out std_logic;
            o_busy : out std_logic;
            o_harm_array_real : out HARM_ARRAY_T;
            o_harm_array_imag : out HARM_ARRAY_T;
            o_dc_samples : out signed(17 downto 0);
            o_ms_samples : out unsigned(25 downto 0)
            );
        end component;


        -- post processing
        component raft_correction
        port(
            i_clk : in std_logic;
            i_na_reset : in std_logic;
            i_busy : in std_logic;
            i_ack_data : in std_logic;
            i_harm_array_real : in HARM_ARRAY_T;
            i_harm_array_imag : in HARM_ARRAY_T;
            i_ms_samples : in unsigned(25 downto 0);
            i_dc_samples : in signed(17 downto 0);
            o_done : out std_logic;
            o_ack_data : out std_logic;
            o_h1_real_corrected : out signed(17 downto 0);
            o_h1_imag_corrected : out signed(17 downto 0);
            o_ms_power : out unsigned(35 downto 0);
            o_dc_power : out unsigned(35 downto 0);
            o_h1_real_hdi : out signed(17 downto 0);
            o_h1_imag_hdi : out signed(17 downto 0);
            o_p1_real_hdi : out unsigned(35 downto 0);
            o_p1_imag_hdi : out unsigned(35 downto 0);
            o_hd5_numerator : out unsigned(39 downto 0);
            o_hd5_denumerator : out unsigned(39 downto 0)
            );
        end component;

        ------------------------------------------------------------------------
        -- SIGNAL DECLARATIONS
        ------------------------------------------------------------------------
        signal s_busy_from_accu_to_correction        : std_logic;
        signal s_ack_data_from_correction_to_accu    : std_logic;

        signal s_harm_array_real                     : HARM_ARRAY_T;
        signal s_harm_array_imag                     : HARM_ARRAY_T;
        signal s_dc_samples_from_accu_to_correction  : signed(17 downto 0);
        signal s_ms_samples_from_accu_to_correction  : unsigned(25 downto 0);

    ------------------------------------------------------------------------
    -- ARCHITECTURE BEGIN | ARCHITECTURE BEGIN | ARCHITECTURE BEGIN |
    ------------------------------------------------------------------------
    begin

        inst_raft_accumulation_toplevel: raft_accumulation_toplevel port MAP(
            i_clk => i_clk,
            i_na_reset => i_na_reset,
            i_sample => i_sample,
            i_sample_timer_trig => i_sample_timer_trig,
            i_sample_done => i_sample_done,
            i_ack_data => s_ack_data_from_correction_to_accu,
            o_sample_start => o_sample_start,
            o_busy => s_busy_from_accu_to_correction,
            o_harm_array_real => s_harm_array_real,
            o_harm_array_imag => s_harm_array_imag,
            o_dc_samples => s_dc_samples_from_accu_to_correction,
            o_ms_samples => s_ms_samples_from_accu_to_correction
        );

        inst_raft_correction: raft_correction port MAP(
            i_clk => i_clk,
            i_na_reset => i_na_reset,
            i_busy => s_busy_from_accu_to_correction,
            i_ack_data => i_ack_data,
            i_harm_array_real => s_harm_array_real,
            i_harm_array_imag => s_harm_array_imag,
            i_ms_samples => s_ms_samples_from_accu_to_correction,
            i_dc_samples => s_dc_samples_from_accu_to_correction,
            o_done => o_done,
            o_ack_data => s_ack_data_from_correction_to_accu,
            o_h1_real_corrected => o_h1_real_corrected,
            o_h1_imag_corrected => o_h1_imag_corrected,
            o_ms_power => o_ms_power,
```

```
            o_dc_power => o_dc_power,
            o_h1_real_hdi => o_h1_real_hdi, -- debug only
            o_h1_imag_hdi => o_h1_imag_hdi, -- debug only
146         o_p1_real_hdi => o_p1_real_hdi,
            o_p1_imag_hdi => o_p1_imag_hdi,
            o_hd5_numerator => o_hd5_numerator,
            o_hd5_denumerator => o_hd5_denumerator
        );
151
    end architecture;
    ------------------------------------------------------------------------------
    -- ARCHITECTURE END  |  ARCHITECTURE END  |  ARCHITECTURE END  |
    ------------------------------------------------------------------------------
```

Listing C.1: raft_toplevel.vhd - Das Toplevel der RAFT-Implementierung mit Instanziierung des Pre- und Postprocessings

```
--------------------------------------------------------------------------------
-- Entity: raft_toplevel_tb
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename            : raft_toplevel_tb.vhd
-- Creation date       : 2013-01-03
-- Author(s)           : Fabian Zahn
-- Version             : 1.00
-- Description         : TESTBENCH RAFT TOPLEVEL
--------------------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
use work.raft_pkg.all;

use STD.textio.all;

ENTITY raft_toplevel_tb IS
end raft_toplevel_tb;

ARCHITECTURE behavior OF raft_toplevel_tb IS

    -- component Declaration for the Unit Under Test (UUT)

    component raft_toplevel
    port(
         i_clk : IN  std_logic;
         i_na_reset : IN  std_logic;
         i_sample : IN  signed(9 downto 0);
         i_sample_timer_trig : IN  std_logic;
         i_sample_done : IN  std_logic;
         i_ack_data : IN  std_logic;
         o_done : OUT  std_logic;
         o_sample_start : OUT  std_logic;
         o_h1_real_corrected : OUT  signed(17 downto 0);
         o_h1_imag_corrected : OUT  signed(17 downto 0);
         o_h1_real_hdi : OUT  signed(17 downto 0);
         o_h1_imag_hdi : OUT  signed(17 downto 0);
         o_ms_power : OUT  unsigned(35 downto 0);
         o_dc_power : OUT  unsigned(35 downto 0);
         o_p1_real_hdi : OUT  unsigned(35 downto 0);
         o_p1_imag_hdi : OUT  unsigned(35 downto 0);
         o_hd5_numerator : OUT  unsigned(39 downto 0);
         o_hd5_denumerator : OUT  unsigned(39 downto 0)
        );
    end component;


   --Inputs
    signal i_clk : std_logic := '0';
    signal i_na_reset : std_logic := '0';
    signal i_sample : signed(9 downto 0) := (others => '0');
    signal i_sample_timer_trig : std_logic := '0';
    signal i_sample_done : std_logic := '0';
    signal i_ack_data : std_logic := '0';

    --Outputs
    signal o_done : std_logic;
    signal o_sample_start : std_logic;
    signal o_h1_real_corrected : signed(17 downto 0);
    signal o_h1_imag_corrected : signed(17 downto 0);
    signal o_h1_real_hdi : signed(17 downto 0);
    signal o_h1_imag_hdi : signed(17 downto 0);
    signal o_ms_power : unsigned(35 downto 0);
    signal o_dc_power : unsigned(35 downto 0);
    signal o_p1_real_hdi : unsigned(35 downto 0);
    signal o_p1_imag_hdi : unsigned(35 downto 0);
    signal o_hd5_numerator : unsigned(39 downto 0);
    signal o_hd5_denumerator : unsigned(39 downto 0);

   -- Clock period definitions
    constant i_clk_period : time := 10 ns;
    constant OMEGA : real := ieee.math_real.math_2_pi/real(240);

   -- file i/o
    constant filename : string :=  "d:\\inputdata.txt";
    file infile : text;

begin

    -- Instantiate the Unit Under Test (UUT)
    uut: raft_toplevel port map (
```

```
            i_clk => i_clk,
            i_na_reset => i_na_reset,
            i_sample => i_sample,
            i_sample_timer_trig => i_sample_timer_trig,
            i_sample_done => i_sample_done,
            i_ack_data => i_ack_data,
            o_done => o_done,
            o_sample_start => o_sample_start,
            o_h1_real_corrected => o_h1_real_corrected,
            o_h1_imag_corrected => o_h1_imag_corrected,
            o_h1_real_hdi => o_h1_real_hdi,
            o_h1_imag_hdi => o_h1_imag_hdi,
            o_ms_power => o_ms_power,
            o_dc_power => o_dc_power,
            o_p1_real_hdi => o_p1_real_hdi,
            o_p1_imag_hdi => o_p1_imag_hdi,
            o_hd5_numerator => o_hd5_numerator,
            o_hd5_denumerator => o_hd5_denumerator
        );

    -- Clock process definitions
    i_clk_process :process
    begin
        i_clk <= '0';
        wait for i_clk_period/2;
        i_clk <= '1';
        wait for i_clk_period/2;
    end process;


    -- Stimulus process
    stim_proc: process
        variable fileline : line;
        variable samplevalue : integer := 0;
    begin
        file_open(infile, filename, READ_MODE);
        -- hold reset state for 100 ns.
        wait for 100 ns;

        ------------------
        -- TEST CASE #1 --
        ------------------
        -- dc and ms test
        i_sample <= to_signed(511, 10);
        i_na_reset <= '1';
        i_sample_timer_trig <= '1'; wait for 15 ns;i_sample_timer_trig <= '0'; -- trigger
        i_sample_done <= '1';
        wait until o_done = '1';
        -- dc and ms assertions
        assert o_dc_power = "00111000000001111100111000010000000" report "ERROR: dc doesn't match" severity ...
            error; -- (511*240)^2
        assert o_ms_power = "00111000000001111100111000010000000" report "ERROR: ms doesn't match" severity ...
            error; -- (511*240)^2


        ------------------
        -- TEST CASE #2 --
        ------------------
        i_sample_done <= '0';
        wait for 500 ns;
        i_ack_data <= '1'; wait for 15 ns; i_ack_data <= '0';
        wait for 1000 ns;

        --i_na_reset <= '0'; wait for 15 ns; i_na_reset <= '1';
        i_sample_timer_trig <= '1'; wait for 15 ns;i_sample_timer_trig <= '0'; -- trigger
        i_sample <= to_signed(0, i_sample'length);

        i_sample_done <= '1';
        for i in 1 to 240-1 loop
            wait until o_sample_start = '1';


            -- file i/o
            readline(infile, fileline);
            read(fileline, samplevalue);
            -- write file input to sample
            i_sample <= to_signed(samplevalue, i_sample'length);

            --i_sample <= to_signed(i,i_sample'length);
            --i_sample <= to_signed(
            -- 2 period sampling
            --integer(ieee.math_real.sin(real(2*i) * omega) * real(511))
            --, i_sample'length
            --);
```

```
170             wait until o_sample_start = '0';
            end loop;

            wait until o_done = '1';

175         ------------------
            -- TEST CASE #3 --
            ------------------
            -- ack data -> go on next test case
            i_sample <= to_signed(0, 10); -- set input sample to zero
180         i_sample_done <= '0';
            wait for 500 ns;
            i_ack_data <= '1'; wait for 15 ns; i_ack_data <= '0';
            wait for 1000 ns;
            -- nothing here yet
185
        wait;
    end process;

end;
```

Listing C.2: raft_toplevel_tb.vhd - Testbench des Toplevels

C.2. Toplevel Akkumulation

```
1  ----------------------------------------------------------------------
   -- Entity: raft_accumulation_toplevel
   ----------------------------------------------------------------------
   -- Copyright 2012
   -- Filename           : raft_accumulation_toplevel.vhd
6  -- Creation date      : 2012-11-08
   -- Author(s)          : Fabian Zahn
   -- Version            : 1.00
   -- Description        : Implementation of the raft accumulation tl
   ----------------------------------------------------------------------
11 -- File History:
   -- Date          Version   Author    Comment
   ----------------------------------------------------------------------
   library ieee;
   use ieee.std_logic_1164.ALL;
16 use ieee.numeric_std.ALL;
   use work.raft_pkg.all;

   entity raft_accumulation_toplevel is
       port(
21             i_clk                : in std_logic;
               i_na_reset           : in std_logic;
               i_sample             : in signed(9 downto 0);
               i_sample_timer_trig  : in std_logic;
               i_sample_done        : in std_logic;
26             i_ack_data           : in std_logic;
               o_sample_start       : out std_logic;
               o_busy               : out std_logic;
               o_harm_array_real    : out HARM_ARRAY_T;
               o_harm_array_imag    : out HARM_ARRAY_T;
31             o_dc_samples         : out signed(17 downto 0);
               o_ms_samples         : out unsigned(25 downto 0)
           );
   end entity;

36
   architecture Behavioral of raft_accumulation_toplevel is

       ----------------------------------------------------------------------
       -- COMPONENT DECLARATIONS
41     ----------------------------------------------------------------------

       component lut
       port(
           i_addr : in unsigned(7 downto 0);
46         o_val_sib : out std_logic_vector(4 downto 0);
           o_val_cob : out std_logic_vector(4 downto 0)
           );
       end component;

51     component raft_accumulation_fsm
       port(
           i_clk                : in std_logic;
           i_na_reset           : in std_logic;
           i_sample_timer_trig  : in std_logic;
56         i_sample_done        : in std_logic;
           i_ack_data           : in std_logic;
           o_clear              : out std_logic;
           o_busy               : out std_logic;
           o_sample_start       : out std_logic;
61         o_en_real            : out std_logic;
           o_en_imag            : out std_logic;
           o_en_dc_ms           : out std_logic;
           o_en_cob             : out std_logic;
           o_lut_address        : out unsigned(7 downto 0)
66         );
       end component;

       component raft_output_reg
       port(
71         i_clk            : in std_logic;
           i_na_reset       : in std_logic;
           i_clear          : in std_logic;
           i_en_real        : in std_logic;
           i_en_imag        : in std_logic;
76         i_en_dc_samples  : in std_logic;
           i_en_ms_samples  : in std_logic;
           i_harm_real      : in HARM_ARRAY_T;
           i_harm_imag      : in HARM_ARRAY_T;
```

```
            i_dc_samples     : in signed(17 downto 0);
            i_ms_samples     : in unsigned(25 downto 0);
            o_harm_real      : out HARM_ARRAY_T;
            o_harm_imag      : out HARM_ARRAY_T;
            o_dc_samples     : out signed(17 downto 0);
            o_ms_samples     : out unsigned(25 downto 0)
            );
        end component;

        component raft_alu_top
        port(
            i_sample             : in signed(9 downto 0);
            i_select_sub_add     : in std_logic_vector(4 downto 0);
            i_harm_array         : in HARM_ARRAY_T;
            i_dc_samples         : in signed(17 downto 0);
            i_ms_samples         : in unsigned(25 downto 0);
            o_harm_array         : out HARM_ARRAY_T;
            o_dc_samples         : out signed(17 downto 0);
            o_ms_samples         : out unsigned(25 downto 0)
            );
        end component;

        ------------------------------------------------------------------------
        -- SIGNAL DECLARATIONS
        ------------------------------------------------------------------------

        signal s_lut_address     : unsigned(7 downto 0);
        signal s_sub_add         : std_logic_vector(4 downto 0);
        signal s_cob             : std_logic_vector(4 downto 0);
        signal s_sib             : std_logic_vector(4 downto 0);

        signal s_en_cob          : std_logic;

        signal s_en_real         : std_logic;
        signal s_en_imag         : std_logic;
        signal s_en_dc_ms        : std_logic;

        signal s_sample_start    : std_logic;
        signal s_clear           : std_logic;
        signal s_reg_clear       : std_logic;

        signal s_harm_array_in   : HARM_ARRAY_T;
        signal s_harm_array_out  : HARM_ARRAY_T;

        signal s_harm_real_in : HARM_ARRAY_T;
        signal s_harm_imag_in : HARM_ARRAY_T;
        signal s_dc_samples_in : signed(17 downto 0);
        signal s_ms_samples_in : unsigned(25 downto 0);

        signal s_harm_real_out : HARM_ARRAY_T;
        signal s_harm_imag_out : HARM_ARRAY_T;
        signal s_dc_samples_out : signed(17 downto 0);
        signal s_ms_samples_out : unsigned(25 downto 0);

    ------------------------------------------------------------------------
    -- ARCHITECTURE BEGIN | ARCHITECTURE BEGIN | ARCHITECTURE BEGIN |
    ------------------------------------------------------------------------
    begin

        s_reg_clear <= i_na_reset and (not s_clear);

        ------------------------------------------------------------------------
        -- COMPONENT INSTANTIATIONS
        ------------------------------------------------------------------------

        Inst_raft_accumulation_fsm: raft_accumulation_fsm port MAP(
            i_clk => i_clk,
            i_na_reset => i_na_reset,
            i_ack_data => i_ack_data,
            o_busy => o_busy,
            i_sample_timer_trig => i_sample_timer_trig,
            i_sample_done => i_sample_done,
            o_clear => s_clear,
            o_sample_start => s_sample_start,
            o_en_real => s_en_real,
            o_en_imag => s_en_imag,
            o_en_dc_ms => s_en_dc_ms,
            o_en_cob => s_en_cob,
            o_lut_address => s_lut_address
        );

        Inst_lut: lut port MAP(
            i_addr => s_lut_address,
            o_val_sib => s_sib,
            o_val_cob => s_cob
```

```
    );

    Inst_raft_output_reg: raft_output_reg port MAP(
        i_clk => i_clk,
        i_na_reset => s_reg_clear,--i_na_reset, -- spart 25k um^2 -> no muxes needed (s_clear h-active ...
            i_na_reset l-active)
        i_clear => '0',--s_clear,                                   -- see above
        i_en_real => s_en_real,
        i_en_imag => s_en_imag,
        i_en_dc_samples => s_en_dc_ms,
        i_en_ms_samples => s_en_dc_ms,
        -- data in
        i_harm_real => s_harm_real_in,
        i_harm_imag => s_harm_imag_in,
        i_dc_samples => s_dc_samples_in,
        i_ms_samples => s_ms_samples_in,
        -- data out
        o_harm_real => s_harm_real_out,
        o_harm_imag => s_harm_imag_out,
        o_dc_samples => s_dc_samples_out,
        o_ms_samples => s_ms_samples_out
    );

    Inst_raft_alu_top: raft_alu_top port MAP(
        i_sample => i_sample,
        i_select_sub_add => s_sub_add,
        i_harm_array => s_harm_array_in,
        o_harm_array => s_harm_array_out,
        i_dc_samples => s_dc_samples_out,
        o_dc_samples => s_dc_samples_in,
        i_ms_samples => s_ms_samples_out,
        o_ms_samples => s_ms_samples_in
    );

    ------------------------------------------------------------------
    -- MULTIPLEXER IMPLEMENTATION
    ------------------------------------------------------------------

    mux_proc : process(s_en_cob, s_cob, s_sib, s_harm_real_out, s_harm_imag_out)
    begin
        case s_en_cob is
            when '0' =>
                s_sub_add       <= s_sib;
                s_harm_array_in <= s_harm_imag_out;
            when '1' =>
                s_sub_add       <= s_cob;
                s_harm_array_in <= s_harm_real_out;
            when others =>
                s_sub_add       <= s_cob;
                s_harm_array_in <= s_harm_real_out;
        end case;
    end process;


    s_harm_real_in <= s_harm_array_out;
    s_harm_imag_in <= s_harm_array_out;

    ------------------------------------------------------------------
    -- OUTPUT ASSIGNMENTS
    ------------------------------------------------------------------
    o_sample_start <= s_sample_start;

    o_harm_array_real <= s_harm_real_out;
    o_harm_array_imag <= s_harm_imag_out;
    o_dc_samples <= s_dc_samples_out;
    o_ms_samples <= s_ms_samples_out;

end architecture;
------------------------------------------------------------------
-- ARCHITECTURE END | ARCHITECTURE END | ARCHITECTURE END |
------------------------------------------------------------------
```

Listing C.3: raft_accumulation_toplevel.vhd - Das RAFT-Preprocessing: die gesteuerte Akkumulation der Abtastwerte

```
1  ----------------------------------------------------------------------
   -- Entity: raft_accumulation_toplevel_tb
   ----------------------------------------------------------------------
   -- Copyright 2012
   -- Filename           : raft_toplevel_tb.vhd
6  -- Creation date      : 2012-11-08
   -- Author(s)          : Fabian Zahn
   -- Version            : 1.00
   -- Description        : TESTBENCH RAFT TOPLEVEL
   ----------------------------------------------------------------------
11 -- File History:
   -- Date          Version  Author   Comment
   ----------------------------------------------------------------------
   LIBRARY ieee;
   USE ieee.std_logic_1164.ALL;
16 USE ieee.numeric_std.ALL;
   use work.raft_pkg.all;

   ENTITY raft_accumulation_toplevel_tb IS
   end ENTITY;
21
   ARCHITECTURE behavior OF raft_accumulation_toplevel_tb IS

       -- component Declaration for the Unit Under Test (UUT)

26     component raft_accumulation_toplevel
       PORT(
            i_clk                  : in  std_logic;
            i_na_reset             : in  std_logic;
            i_sample               : in  signed(9 downto 0);
31          i_sample_timer_trig    : in  std_logic;
            i_sample_done          : in  std_logic;
            i_ack_data             : in std_logic;
            o_sample_start         : out  std_logic;
            o_busy                 : out std_logic;
36          o_harm_array_real      : out  HARM_ARRAY_T;
            o_harm_array_imag      : out  HARM_ARRAY_T;
            o_dc_samples           : out  signed(17 downto 0);
            o_ms_samples           : out  unsigned(25 downto 0)
           );
41     end component;


      --inputs
       signal i_clk : std_logic := '0';
46     signal i_na_reset : std_logic := '0';
       signal i_sample : signed(9 downto 0) := (others => '0');
       signal i_sample_timer_trig : std_logic := '0';
       signal i_sample_done : std_logic := '0';
       signal i_ack_data : std_logic := '0';
51
        --outputs
       signal o_sample_start : std_logic;
       signal o_harm_array_real : HARM_ARRAY_T;
       signal o_harm_array_imag : HARM_ARRAY_T;
56     signal o_dc_samples : signed(17 downto 0);
       signal o_ms_samples : unsigned(25 downto 0);
       signal o_busy : std_logic;

      -- Clock period definitions
61     constant i_clk_period : time := 10 ns;
       constant OMEGA : real := ieee.math_real.math_2_pi/real(240);

   BEGin

66      -- instantiate the Unit Under Test (UUT)
       uut: raft_accumulation_toplevel PORT MAP (
              i_clk => i_clk,
              i_na_reset => i_na_reset,
              i_sample => i_sample,
71            i_sample_timer_trig => i_sample_timer_trig,
              i_sample_done => i_sample_done,
              i_ack_data => i_ack_data,
              o_busy => o_busy,
              o_sample_start => o_sample_start,
76            o_harm_array_real => o_harm_array_real,
              o_harm_array_imag => o_harm_array_imag,
              o_dc_samples => o_dc_samples,
              o_ms_samples => o_ms_samples
            );
81
      -- Clock process definitions
       i_clk_process :process
       begin
            i_clk <= '0';
```

```
86          wait for i_clk_period/2;
            i_clk <= '1';
            wait for i_clk_period/2;
        end process;

91
        -- Stimulus process
        stim_proc: process
        begin
           -- hold reset state for 100 ns.
96          wait for 100 ns;

            ------------------
            -- TEST CASE #1 --
            ------------------
101         -- dc and ms test
            i_sample <= to_signed(511, 10);
            i_na_reset <= '1';
            i_sample_timer_trig <= '1'; wait for 15 ns;i_sample_timer_trig <= '0'; -- trigger
            i_sample_done <= '1';
106         wait until o_busy = '0';
            -- dc and ms assertions
            assert o_dc_samples = to_signed(511*240, o_dc_samples'length) report "ERROR: dc doesn't match" ...
                severity error;
            assert o_ms_samples = to_unsigned(511*511*240, o_ms_samples'length) report "ERROR: ms doesn't match" ...
                severity error;
            wait for 100 ns;
111
            ------------------
            -- TEST CASE #2 --
            ------------------
            -- sine test
116         i_sample_done <= '0';
            wait for 500 ns;
            i_ack_data <= '1'; wait for 15 ns; i_ack_data <= '0';
            wait for 1000 ns;

121         i_sample_timer_trig <= '1'; wait for 15 ns;i_sample_timer_trig <= '0'; -- trigger
            i_sample <= to_signed(0, i_sample'length);

            i_sample_done <= '1';
            for i in 1 to 240-1 loop
126             wait until o_sample_start = '1';

                --i_sample <= to_signed(i,i_sample'length);
                i_sample <= to_signed(
                            integer(ieee.math_real.sin(real(i) * omega) * real(511))
131                         , i_sample'length
                            );


                wait until o_sample_start = '0';
136         end loop;

          wait;
        end process;
    end;
```

Listing C.4: raft_accumulation_toplevel_tb.vhd - Testbench der RAFT-Akkumulation

C.3. Zustandsautomat: Akkumulation

```
--------------------------------------------------------------------------------
-- Entity: raft_accumulation_fsm
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename            : raft_accumulation_fsm.vhd
-- Creation date       : 2012-11-20
-- Author(s)           : Fabian Zahn
-- Version             : 1.00
-- Description         : Implementation of the raft calculation
--                       fsm (including the internal sample counter
--------------------------------------------------------------------------------
-- File History:
-- Date          Version   Author    Comment
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.raft_pkg.all;

entity raft_accumulation_fsm is
    port(
            ----------------------------------------------------------------
            -- state machine
                i_clk               : in std_logic;
                i_na_reset          : in std_logic;

            ----------------------------------------------------------------
            -- correction fsm
                i_ack_data          : in std_logic;
                o_busy              : out std_logic;

            ----------------------------------------------------------------
            -- sample logic
                i_sample_timer_trig : in std_logic;
                i_sample_done       : in std_logic;
                o_sample_start      : out std_logic;

            ----------------------------------------------------------------
            -- registers
                o_clear             : out std_logic;
                -- enable signals
                o_en_real           : out std_logic;
                o_en_imag           : out std_logic;

                -- registers needed for hdi calculation
                o_en_dc_ms          : out std_logic;

            ----------------------------------------------------------------
            -- lut
                o_en_cob        : out std_logic;
                o_lut_address   : out unsigned(7 downto 0)

        );
end entity;

architecture Behavioral of raft_accumulation_fsm is

    -- RAFT FSM STATES
    type RAFT_FSM_STATE_T is (
        -- INITIAL STATE
        INIT_S,
        -- RAFT ACCUMULATION
        IDLE_S,
        ADD_REAL_S,
        ADD_IMAG_S,
        INCREMENT_SAMPLE_CTR_S,
        DONE_S
    );

    signal s_current_state, s_next_state : RAFT_FSM_STATE_T;

    signal s_counter_state : unsigned(7 downto 0);
    signal s_en_sample_ctr : std_logic;
    signal s_reset_ctr     : std_logic;

begin
    -- SAMPLE COUNTER
    sample_ctr_p : process(i_clk, i_na_reset)
```

```
80      begin
            if i_na_reset = '0' then
                s_counter_state <= (others => '0');
            elsif rising_edge(i_clk) then
                if s_reset_ctr = '1' then
85                  s_counter_state <= (others => '0');
                else
                    if s_en_sample_ctr = '1' then
                        s_counter_state <= s_counter_state + 1;
                    end if;
90              end if;
            end if;
        end process;


95      -- STATE MACHINE REGISTER
        state_reg_p : process (i_clk, i_na_reset)
        begin
            if i_na_reset = '0' then
                s_current_state <= INIT_S;
100         elsif rising_edge(i_clk) then
                s_current_state <= s_next_state;
            end if;
        end process;

105
        -- STATE MACHINE COMBINATORICAL LOGIC
        state_machine_comb_p : process(s_current_state, s_counter_state, i_sample_timer_trig, i_sample_done, i_ack_data)
        begin
            ------------------------------
110         -- begin : default assignments
            ------------------------------
            s_next_state        <= s_current_state;
            s_en_sample_ctr     <= '0';
            s_reset_ctr         <= '0';
115         o_sample_start      <= '0';

            o_en_real           <= '0';
            o_en_imag           <= '0';
            o_en_dc_ms          <= '0';
120
            o_en_cob            <= '1';
            o_clear             <= '0';
            o_busy              <= '1';

125         ------------------------------
            -- end : default assignments
            ------------------------------


            ------------------------------------
130         -- begin: next state and output logic
            ------------------------------------
            case s_current_state is
                when INIT_S =>
                    if i_sample_timer_trig = '1' then
135                     s_next_state <= IDLE_S;
                    end if;
                    s_reset_ctr <= '1';
                    o_clear <= '1';
                    --o_busy <= '0';
140
                when IDLE_S =>
                    o_sample_start <= '1'; -- request sample
                    if i_sample_done = '1' then
                        s_next_state <= ADD_REAL_S;
145                 end if;

                when ADD_REAL_S =>
                    s_next_state <= ADD_IMAG_S;
                    o_en_dc_ms <= '1';
150                 o_en_real <= '1';


                when ADD_IMAG_S =>
                    s_next_state <= INCREMENT_SAMPLE_CTR_S;
155                 o_en_imag <= '1';
                    o_en_cob  <= '0';

                when INCREMENT_SAMPLE_CTR_S =>
                    s_en_sample_ctr <= '1';
160                 if s_counter_state < 239 then
                        s_next_state <= IDLE_S;
                    else
                        s_next_state <= DONE_S;
                    end if;
```

```
165
                when DONE_S => -- 240 samples collected
                    o_busy <= '0'; -- the machine seems always to be busy to this is signal is used by the ...
                        correction fsm
                    -- go to init when the registers can be rewritten again (correction is done)
                    if i_ack_data = '1' then
170                     s_next_state <= INIT_S;
                    end if;

            end case;
            ---------------------------------------
175         -- end : next state and output logic
            ---------------------------------------

        end process; -- end state machine comb. logic

180     ---------------------------------
        -- unconditional signal assignments
        ---------------------------------
        o_lut_address <= s_counter_state;

185 end architecture;
```

Listing C.5: raft_accumulation_fsm.vhd - Zustandsautomat für die Steuerung der Akkumulation von Abtastwerten

C.4. Toplevel Akkumulations ALU

```
--------------------------------------------------------------------------------
-- Entity: raft_alu
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename            : raft_alu_top.vhd
-- Creation date       : 2012-11-08
-- Author(s)           : Fabian Zahn
-- Version             : 1.00
-- Description         : Implementation of the raft alu toplevel
--                       for all kinds of calculation
--------------------------------------------------------------------------------
-- File History:
-- Date          Version   Author    Comment
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
use work.raft_pkg.all;

entity raft_alu_top is
    port(
            -- general
            i_sample            : in signed (9 downto 0);   -- input sample 10 bit

            i_select_sub_add    : in std_logic_vector(4 downto 0);
            i_harm_array        : in HARM_ARRAY_T;          -- for real or imaginary input
            o_harm_array        : out HARM_ARRAY_T;

            -- dc sample adder / correction adder
            i_dc_samples          : in  signed (17 downto 0);
            o_dc_samples          : out signed (17 downto 0);   -- adder output (used for dc estimation)

            -- ms sample calc
            i_ms_samples          : in  unsigned(25 downto 0);
            o_ms_samples          : out unsigned(25 downto 0)
        );
end entity;

architecture Behavioral of raft_alu_top is
    -- components
    component raft_alu
        port (  i_mode_select       : in  std_logic;
                i_sample            : in  signed (9 downto 0);
                i_accumulator_reg   : in  signed (17 downto 0);
                o_result            : out signed (17 downto 0)
             );
    end component;
    -- signals
    signal s_product : signed (19 downto 0);
    signal s_product_unsigned : unsigned (19 downto 0);

begin
    -- 5 instances of raft_alu used for real and imaginary part
    raft_alu_i: for k in 0 to 4 generate
    begin
        raft_alu_ix : raft_alu
            port map(
                i_mode_select       => i_select_sub_add(k),
                i_sample            => i_sample,
                i_accumulator_reg   => i_harm_array(k),
                o_result            => o_harm_array(k)
            );
    end generate;

    -- 10 bit unsigned multiplier for ms estimation
    s_product <= i_sample * i_sample;
    s_product_unsigned <= to_unsigned(to_integer(s_product), s_product_unsigned'length);
    o_ms_samples <= i_ms_samples + resize(s_product_unsigned, i_ms_samples'length);

    -- and finally one adder for the dc power estimation
    o_dc_samples <= i_dc_samples + resize(i_sample, i_dc_samples'length);

end architecture;
```

Listing C.6: raft_alu_top.vhd - Akkumulations ALU bestehend aus Addierern und Subtrahierern für die Abtastwerte

```
--------------------------------------------------------------------------------
-- Entity: raft_alu
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename            : raft_alu.vhd
-- Creation date       : 2012-11-08
-- Author(s)           : Fabian Zahn
-- Version             : 1.00
-- Description         : Implementation of the raft alu
--                       (adder / sub) for one harm
--------------------------------------------------------------------------------
-- File History:
-- Date          Version   Author    Comment
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;

entity raft_alu is
    Port (  i_mode_select       : in  std_logic;
            i_sample            : in  signed (9 downto 0);
            i_accumulator_reg   : in  signed (17 downto 0);
            o_result            : out signed (17 downto 0)
         );
end entity;

architecture Behavioral of raft_alu is

begin

    process (i_mode_select, i_sample, i_accumulator_reg)
    begin
        if i_mode_select = '1' then
            o_result <= i_accumulator_reg + resize(i_sample, i_accumulator_reg'length);
        elsif i_mode_select = '0' then
            o_result <= i_accumulator_reg - resize(i_sample, i_accumulator_reg'length);
        end if;
    end process;

end architecture;
```

Listing C.7: raft_alu.vhd - Testbench der RAFT-Akkumulations ALU

C.5. Akkumulationsregister

```
--------------------------------------------------------------------------------
-- Entity: raft_output_reg
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename           : raft_output_reg.vhd
-- Creation date      : 2012-11-08
-- Author(s)          : Fabian Zahn
-- Version            : 1.00
-- Description        : Output registers used by the raft calculation
--------------------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.raft_pkg.all;

entity raft_output_reg is
    port(
            i_clk           : in std_logic;
            i_na_reset      : in std_logic;
            i_clear         : in std_logic;

            -- enable signals (one hot encoded in order to save an address decoder and to enable more than 1 ...
                reg at once)
            i_en_real           : in std_logic;
            i_en_imag           : in std_logic;

            -- registers needed for hdi calculation
            i_en_dc_samples     : in std_logic;
            i_en_ms_samples     : in std_logic;


            -- data in
            i_harm_real     : in HARM_ARRAY_T;
            i_harm_imag     : in HARM_ARRAY_T;
            i_dc_samples    : in signed (17 downto 0);
            i_ms_samples    : in unsigned (25 downto 0);

            -- data out
            o_harm_real     : out HARM_ARRAY_T;
            o_harm_imag     : out HARM_ARRAY_T;
            o_dc_samples    : out signed (17 downto 0);
            o_ms_samples    : out unsigned (25 downto 0)

        );
end raft_output_reg;

architecture behavioural of raft_output_reg is
    -- complex amplitudes
    signal s_harm_real      : HARM_ARRAY_T;
    signal s_harm_imag      : HARM_ARRAY_T;
    -- dc and ms power
    signal s_dc_samples     : signed (17 downto 0);
    signal s_ms_samples     : unsigned (25 downto 0);
begin

    ----------------------------------------------------------------------------
    -- register process
    reg_p : process(i_clk, i_na_reset)
    begin
        -- low active reset (asynchronous)
        if i_na_reset = '0' then

            for i in 0 to (s_harm_real'length-1) loop
                s_harm_real(i) <= (others => '0');
            end loop;

            for i in 0 to (s_harm_imag'length-1) loop
                s_harm_imag(i) <= (others => '0');
            end loop;

            s_dc_samples <= (others => '0');
            s_ms_samples <= (others => '0');

        -- clocked part
        elsif rising_edge(i_clk) then
```

```
                -- real part of the harmonics
                for i in 0 to (s_harm_real'length-1) loop
                    if i_clear = '1' then
                        s_harm_real(i) <= (others => '0');
                    elsif i_en_real = '1' then
                        s_harm_real(i) <= i_harm_real(i);
                    end if;
                end loop;

                -- imag part of the harmonics
                for i in 0 to (s_harm_imag'length-1) loop
                    if i_clear = '1' then
                        s_harm_imag(i) <= (others => '0');
                    elsif i_en_imag = '1' then
                        s_harm_imag(i) <= i_harm_imag(i);
                    end if;
                end loop;

                -- dc samples
                if i_clear = '1' then
                    s_dc_samples    <= (others => '0');
                elsif i_en_dc_samples = '1' then
                    s_dc_samples    <= i_dc_samples;
                end if;

                -- dc power
                if i_clear = '1' then
                    s_ms_samples <= (others => '0');
                elsif i_en_ms_samples = '1' then
                    s_ms_samples    <= i_ms_samples;
                end if;

            end if;
        end process;

        ------------------------------------------------------------
        -- output assignments
        o_harm_real     <= s_harm_real;
        o_harm_imag     <= s_harm_imag;
        o_dc_samples    <= s_dc_samples;
        o_ms_samples    <= s_ms_samples;

    end behavioural;
```

Listing C.8: raft_output_reg.vhd - Register für die Speicherung der Akkumulationsergebnisse

C.6. Lookup-Table

```
--------------------------------------------------------------------------------
-- Entity: lut
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename            : lut.vhd
-- Creation date       : 2012-06-05
-- Author(s)           : Fabian Zahn
-- Version             : 1.00
-- Description         : rademacher mux signal generation using
--                       a look-up-table or rom (depending on synthesis)
--------------------------------------------------------------------------------
-- File History:
-- Date          Version   Author    Comment
--------------------------------------------------------------------------------

-- INCLUDE
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- ENTITY
--
entity lut is
    port(
            i_addr    : in unsigned(7 downto 0);
            o_val_sib : out std_logic_vector(4 downto 0);
            o_val_cob : out std_logic_vector(4 downto 0)
    );
end entity;


--
-- ARCHITECTURE
--
architecture arch of lut is

    -- generated arrays
    type array_t is array ( 0 to 255 ) of std_logic_vector(4 downto 0);
    constant SIB_TAB : array_t := (0 => "00000",1 => "00000",2 => "00000",
        3 => "00000",4 => "00000",5 => "00000",6 => "00000",7 => "00000",
        8 => "00000",9 => "00000",10 => "00000",11 => "00000",12 => "00000",
        13 => "00000",14 => "00000",15 => "00000",16 => "00000",17 => "00000",
        18 => "00000",19 => "00000",20 => "00000",21 => "00000",22 => "00000",
        23 => "00000",24 => "10000",25 => "10000",26 => "10000",27 => "10000",
        28 => "10000",29 => "10000",30 => "11000",31 => "11000",32 => "11000",
        33 => "11000",34 => "11000",35 => "11000",36 => "11000",37 => "11000",
        38 => "11000",39 => "11000",40 => "11100",41 => "11100",42 => "11100",
        43 => "11100",44 => "11100",45 => "11100",46 => "11100",47 => "11100",
        48 => "01100",49 => "01100",50 => "01100",51 => "01100",52 => "01100",
        53 => "01100",54 => "01100",55 => "01100",56 => "01100",57 => "01100",
        58 => "01100",59 => "01100",60 => "00110",61 => "00110",62 => "00110",
        63 => "00110",64 => "00110",65 => "00110",66 => "00110",67 => "00110",
        68 => "00110",69 => "00110",70 => "00110",71 => "00110",72 => "10110",
        73 => "10110",74 => "10110",75 => "10110",76 => "10110",77 => "10110",
        78 => "10110",79 => "10110",80 => "10010",81 => "10010",82 => "10010",
        83 => "10010",84 => "10010",85 => "10010",86 => "10010",87 => "10010",
        88 => "10010",89 => "10010",90 => "11010",91 => "11010",92 => "11010",
        93 => "11010",94 => "11010",95 => "11010",96 => "01010",97 => "01010",
        98 => "01010",99 => "01010",100 => "01010",101 => "01010",102 => "01010",
        103 => "01010",104 => "01010",105 => "01010",106 => "01010",107 => "01010",
        108 => "01010",109 => "01010",110 => "01010",111 => "01010",112 => "01010",
        113 => "01010",114 => "01010",115 => "01010",116 => "01010",117 => "01010",
        118 => "01010",119 => "01010",120 => "10101",121 => "10101",122 => "10101",
        123 => "10101",124 => "10101",125 => "10101",126 => "10101",127 => "10101",
        128 => "10101",129 => "10101",130 => "10101",131 => "10101",132 => "10101",
        133 => "10101",134 => "10101",135 => "10101",136 => "10101",137 => "10101",
        138 => "10101",139 => "10101",140 => "10101",141 => "10101",142 => "10101",
        143 => "10101",144 => "00101",145 => "00101",146 => "00101",147 => "00101",
        148 => "00101",149 => "00101",150 => "01101",151 => "01101",152 => "01101",
        153 => "01101",154 => "01101",155 => "01101",156 => "01101",157 => "01101",
        158 => "01101",159 => "01101",160 => "01001",161 => "01001",162 => "01001",
        163 => "01001",164 => "01001",165 => "01001",166 => "01001",167 => "01001",
        168 => "11001",169 => "11001",170 => "11001",171 => "11001",172 => "11001",
        173 => "11001",174 => "11001",175 => "11001",176 => "11001",177 => "11001",
        178 => "11001",179 => "11001",180 => "10011",181 => "10011",182 => "10011",1
        83 => "10011",184 => "10011",185 => "10011",186 => "10011",187 => "10011",
        188 => "10011",189 => "10011",190 => "10011",191 => "10011",192 => "00011",
        193 => "00011",194 => "00011",195 => "00011",196 => "00011",197 => "00011",
```

```
            198 => "00011",199 => "00011",200 => "00111",201 => "00111",202 => "00111",
            203 => "00111",204 => "00111",205 => "00111",206 => "00111",207 => "00111",
            208 => "00111",209 => "00111",210 => "01111",211 => "01111",212 => "01111",
            213 => "01111",214 => "01111",215 => "01111",216 => "11111",217 => "11111",
            218 => "11111",219 => "11111",220 => "11111",221 => "11111",222 => "11111",
            223 => "11111",224 => "11111",225 => "11111",226 => "11111",227 => "11111",
            228 => "11111",229 => "11111",230 => "11111",231 => "11111",232 => "11111",
            233 => "11111",234 => "11111",235 => "11111",236 => "11111",237 => "11111",
            238 => "11111",239 => "11111",240 => "------", 241 => "------", 242 => "------",
            243 => "------", 244 => "------", 245 => "------", 246 => "------", 247 => "------",
            248 => "------", 249 => "------", 250 => "------", 251 => "------", 252 => "------",
            253 => "------", 254 => "------", 255 => "------");

    constant COB_TAB : array_t := (0 => "11111",1 => "11111",2 => "11111",3 => "11111",
        4 => "11111",5 => "11111",6 => "11111",7 => "11111",8 => "11111",9 => "11111",
        10 => "11111",11 => "11111",12 => "01111",13 => "01111",14 => "01111",15 => "00111",
        16 => "00111",17 => "00111",18 => "00111",19 => "00111",20 => "00011",21 => "00011"
        ,22 => "00011",23 => "00011",24 => "00011",25 => "00011",26 => "00011",27 => "00011",
        28 => "00011",29 => "00011",30 => "00001",31 => "00001",32 => "00001",33 => "00001",
        34 => "00001",35 => "00001",36 => "10001",37 => "10001",38 => "10001",39 => "10001",
        40 => "10001",41 => "10001",42 => "10001",43 => "10001",44 => "10001",45 => "11001",
        46 => "11001",47 => "11001",48 => "11001",49 => "11001",50 => "11001",51 => "11001",
        52 => "11001",53 => "11001",54 => "11001",55 => "11001",56 => "11001",57 => "11001",
        58 => "11001",59 => "11001",60 => "01100",61 => "01100",62 => "01100",63 => "01100",
        64 => "01100",65 => "01100",66 => "01100",67 => "01100",68 => "01100",69 => "01100",
        70 => "01100",71 => "01100",72 => "01100",73 => "01100",74 => "01100",75 => "00100",
        76 => "00100",77 => "00100",78 => "00100",79 => "00100",80 => "00100",81 => "00100",
        82 => "00100",83 => "00100",84 => "10100",85 => "10100",86 => "10100",87 => "10100",
        88 => "10100",89 => "10100",90 => "10110",91 => "10110",92 => "10110",93 => "10110",
        94 => "10110",95 => "10110",96 => "10110",97 => "10110",98 => "10110",99 => "10110",
        100 => "10010",101 => "10010",102 => "10010",103 => "10010",104 => "10010",105 => "11010",
        106 => "11010",107 => "11010",108 => "01010",109 => "01010",110 => "01010",111 => "01010",
        112 => "01010",113 => "01010",114 => "01010",115 => "01010",116 => "01010",117 => "01010",
        118 => "01010",119 => "01010",120 => "01010",121 => "01010",122 => "01010",123 => "01010",
        124 => "01010",125 => "01010",126 => "01010",127 => "01010",128 => "01010",129 => "01010",
        130 => "01010",131 => "01010",132 => "11010",133 => "11010",134 => "11010",135 => "10010",
        136 => "10010",137 => "10010",138 => "10010",139 => "10010",140 => "10110",141 => "10110",
        142 => "10110",143 => "10110",144 => "10110",145 => "10110",146 => "10110",147 => "10110",
        148 => "10110",149 => "10110",150 => "10100",151 => "10100",152 => "10100",153 => "10100",
        154 => "10100",155 => "10100",156 => "00100",157 => "00100",158 => "00100",159 => "00100",
        160 => "00100",161 => "00100",162 => "00100",163 => "00100",164 => "00100",165 => "01100",
        166 => "01100",167 => "01100",168 => "01100",169 => "01100",170 => "01100",171 => "01100",
        172 => "01100",173 => "01100",174 => "01100",175 => "01100",176 => "01100",177 => "01100",
        178 => "01100",179 => "01100",180 => "11001",181 => "11001",182 => "11001",183 => "11001",
        184 => "11001",185 => "11001",186 => "11001",187 => "11001",188 => "11001",189 => "11001",
        190 => "11001",191 => "11001",192 => "11001",193 => "11001",194 => "11001",195 => "10001",
        196 => "10001",197 => "10001",198 => "10001",199 => "10001",200 => "10001",201 => "10001",
        202 => "10001",203 => "10001",204 => "00001",205 => "00001",206 => "00001",207 => "00001",
        208 => "00001",209 => "00001",210 => "00011",211 => "00011",212 => "00011",213 => "00011",
        214 => "00011",215 => "00011",216 => "00011",217 => "00011",218 => "00011",219 => "00011",
        220 => "00111",221 => "00111",222 => "00111",223 => "00111",224 => "00111",225 => "01111",
        226 => "01111",227 => "01111",228 => "11111",229 => "11111",230 => "11111",231 => "11111",
        232 => "11111",233 => "11111",234 => "11111",235 => "11111",236 => "11111",237 => "11111",
        238 => "11111",239 => "11111",240 => "------", 241 => "------", 242 => "------", 243 => "------",
         244 => "------", 245 => "------", 246 => "------", 247 => "------", 248 => "------", 249 => "------",
          250 => "------", 251 => "------", 252 => "------", 253 => "------", 254 => "------", 255 => "------");

begin

    o_val_sib <=  SIB_TAB(to_integer(i_addr)) ;
    o_val_cob <=  COB_TAB(to_integer(i_addr)) ;

end architecture;
```

Listing C.9: lut.vhd - Lookup-Table für die cob- bzw. sib-Funktionen (1-Bit)

C.7. Toplevel Postprocessing

```
--------------------------------------------------------------------------------
-- Entity: raft_correction
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename           : raft_correction.vhd
-- Creation date      : 2012-12-15
-- Author(s)          : Fabian Zahn
-- Version            : 1.00
-- Description        : Correction of the harmonics and calculation
--                      of the dc and ms component
--------------------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
use work.raft_pkg.all;

entity raft_correction is
    port(
            --
            -- INPUTS
            --
            i_clk               : in std_logic;
            i_na_reset          : in std_logic;
            i_busy              : in std_logic; -- from previous stage
            i_ack_data          : in std_logic; -- from next stage
            i_harm_array_real   : in HARM_ARRAY_T;
            i_harm_array_imag   : in HARM_ARRAY_T;
            i_ms_samples        : in unsigned(25 downto 0);
            i_dc_samples        : in signed(17 downto 0);
            --
            -- OUTPUTS
            --
            o_done              : out std_logic; -- to next stage
            o_ack_data          : out std_logic; -- to previous stage
            o_h1_real_corrected : out signed(17 downto 0);
            o_h1_imag_corrected : out signed(17 downto 0);
            -- hdi output
            o_ms_power          : out unsigned(35 downto 0);
            o_dc_power          : out unsigned(35 downto 0);
            o_h1_real_hdi       : out signed(17 downto 0); -- for debugging only
            o_h1_imag_hdi       : out signed(17 downto 0); -- for debugging only
            o_p1_real_hdi       : out unsigned(35 downto 0);
            o_p1_imag_hdi       : out unsigned(35 downto 0);
            --hd5 output
            o_hd5_numerator     : out unsigned(39 downto 0);
            o_hd5_denumerator   : out unsigned(39 downto 0)
        );
end entity;

architecture Behavioral of raft_correction is

    --------------------------------------------------------------------------
    -- CONSTANT DECLARATIONS
    --------------------------------------------------------------------------

    -- this constant decides whether we use one or to period sampling
    -- this can be changed into an input signal but this will change
    -- the automata type from moore to mealy
    constant c_en_one_period    : std_logic := '0';

    -- correction constants
    constant ONE_THIRD          : signed(17 downto 0) := to_signed( 171, 18);   -- round( 1/3.0*512) (10 ...
        signed quantisation)
    constant MINUS_ONE_THIRD    : signed(17 downto 0) := to_signed(-171, 18);   -- round(-1/3.0*512) (10 ...
        signed quantisation)
    constant MINUS_ONE_FIFTH    : signed(17 downto 0) := to_signed(-102, 18);   -- round( 1/5.0*512) (10 ...
        signed quantisation)
    constant POWER_SCALE        : signed(17 downto 0) := to_signed( 402, 18);   -- round( math.pi/4*512) (10 ...
        signed quantisation)

    -- mux constants
    constant C_H1_REAL_SEL              : std_logic_vector(3 downto 0) := "0000"; -- 0
    constant C_H2_REAL_SEL              : std_logic_vector(3 downto 0) := "0001"; -- 1
    constant C_H3_REAL_SEL              : std_logic_vector(3 downto 0) := "0010"; -- 2
    constant C_H4_REAL_SEL              : std_logic_vector(3 downto 0) := "0011"; -- 3
    constant C_H5_REAL_SEL              : std_logic_vector(3 downto 0) := "0100"; -- 4
```

```
    constant C_H1_IMAG_SEL              : std_logic_vector(3 downto 0) := "0101"; -- 5
    constant C_H2_IMAG_SEL              : std_logic_vector(3 downto 0) := "0110"; -- 6
    constant C_H3_IMAG_SEL              : std_logic_vector(3 downto 0) := "0111"; -- 7
    constant C_H4_IMAG_SEL              : std_logic_vector(3 downto 0) := "1000"; -- 8
    constant C_H5_IMAG_SEL              : std_logic_vector(3 downto 0) := "1001"; -- 9
    constant C_H1_REAL_CORRECTED_SEL    : std_logic_vector(3 downto 0) := "1010"; -- 10
    constant C_H1_IMAG_CORRECTED_SEL    : std_logic_vector(3 downto 0) := "1011"; -- 11
    constant C_DC_SEL                   : std_logic_vector(3 downto 0) := "1100"; -- 12
    constant C_H1_REAL_HDI_SEL          : std_logic_vector(3 downto 0) := "1101"; -- 13
    constant C_H1_IMAG_HDI_SEL          : std_logic_vector(3 downto 0) := "1110"; -- 14


    ------------------------------------------------------------------------
    -- COMPONENT DECLARATIONS
    ------------------------------------------------------------------------

    -- mac unit
    COMPONENT correction_mac
    PORT(
        in_a : in signed(17 downto 0);
        in_b : in signed(17 downto 0);
        in_c : in signed(17 downto 0);
        i_en_square         : in std_logic;
        in_ms_correction    : in unsigned(25 downto 0);
        o_a_times_b         : out signed(35 downto 0);
        o_ms_times_240      : out unsigned(35 downto 0);
        o_a_times_b_plus_c  : out signed(17 downto 0)
        );
    END COMPONENT;

    ------------------------------------------------------------------------
    -- FSM DECLARATIONS
    ------------------------------------------------------------------------
    type CORRECTION_FSM_STATE_T is(
            IDLE_S,
            INIT_REGS_S,
            REAL_CORRECTION_1_S,
            REAL_CORRECTION_2_S,
            IMAG_CORRECTION_1_S,
            IMAG_CORRECTION_2_S,
            POWER_CORRECTION_REAL_S,
            POWER_CORRECTION_IMAG_S,
            SQUARE_DC_S,
            SQUARE_H1_REAL_HDI_S,
            SQUARE_H1_IMAG_HDI_S,
            ADD_P5_REAL_S,
            ADD_P5_IMAG_S,
            ADD_P4_REAL_S,
            ADD_P4_IMAG_S,
            ADD_P3_REAL_S,
            ADD_P3_IMAG_S,
            ADD_P2_REAL_S,
            ADD_P2_IMAG_S,
            ADD_P1_REAL_S,
            ADD_P1_IMAG_S,
            HANDSHAKE_S,
            DONE_S
    );

    ------------------------------------------------------------------------
    -- SIGNAL DECLARATIONS
    ------------------------------------------------------------------------

    signal s_clear_regs             : std_logic;
    signal s_reg_reset              : std_logic;

    signal s_current_state, s_next_state : CORRECTION_FSM_STATE_T;

    -- MUX signals
    signal s_mux_a_select : std_logic_vector(3 downto 0);
    signal s_mux_b_select : std_logic_vector(1 downto 0);
    signal s_mux_c_select : std_logic;

    -- MAC signals
    signal s_mac_a_in           : signed(17 downto 0);
    signal s_mac_b_in           : signed(17 downto 0);
    signal s_mac_c_in           : signed(17 downto 0);
    signal s_en_square          : std_logic;
    signal s_a_times_b          : signed (35 downto 0);
    signal s_ms_times_240       : unsigned (35 downto 0);
    signal s_a_times_b_plus_c   : signed (17 downto 0);

    --correction regs
    signal s_h1_real_corrected_enable   : std_logic;
    signal s_h1_imag_corrected_enable   : std_logic;
```

```
        signal s_h1_load                    : std_logic;
        signal s_h1_real_corrected  : signed(17 downto 0);
163     signal s_h1_imag_corrected  : signed(17 downto 0);

        -- hdi regs
        signal s_ms_corrected           : unsigned(35 downto 0);
        signal s_dc_squared             : unsigned(35 downto 0);
168     signal s_h1_real_hdi            : signed(17 downto 0);
        signal s_h1_imag_hdi            : signed(17 downto 0);
        signal s_p1_real_hdi            : unsigned(35 downto 0);
        signal s_p1_imag_hdi            : unsigned(35 downto 0);
        -- hdi reg enables
173     signal s_h1_real_hdi_enable     : std_logic;
        signal s_h1_imag_hdi_enable     : std_logic;
        signal s_p1_real_hdi_enable     : std_logic;
        signal s_p1_imag_hdi_enable     : std_logic;
        signal s_dc_squared_enable      : std_logic;
178     signal s_ms_corrected_enable    : std_logic;

        -- hd5 regs
        signal s_hd5_numerator          : unsigned(39 downto 0);
        signal s_hd5_denumerator        : unsigned(39 downto 0);
183     -- hd5 reg enable
        signal s_hd5_numerator_enable   : std_logic;
        signal s_hd5_denumerator_enable   : std_logic;

        signal s_power_output           : unsigned(35 downto 0);
188     signal s_harm_adder             : unsigned(39 downto 0);

    ------------------------------------------------------------------------
    -- ARCHITECTURE BEGIN | ARCHITECTURE BEGIN | ARCHITECTURE BEGIN |
    ------------------------------------------------------------------------
193 begin
        -- asynchronous reset signal generation (needed for the registers)
        s_reg_reset <= i_na_reset and (not s_clear_regs);

        ------------------------------------------------------------------
198     -- MULTIPLEXER IMPLEMENTATIONS
        ------------------------------------------------------------------
        -- muxes h1r h1i; h2r h2i; h3r h3i, h4r h4i; h5r h5i,
        mac_a_mux_p : process(s_mux_a_select, i_harm_array_real, i_harm_array_imag, s_h1_real_corrected, ...
              s_h1_imag_corrected, i_dc_samples, s_h1_real_hdi, s_h1_imag_hdi)
        begin
203         case s_mux_a_select is
                when C_H1_REAL_SEL =>
                    s_mac_a_in <= i_harm_array_real(0);
                when C_H2_REAL_SEL =>
                    s_mac_a_in <= i_harm_array_real(1);
208             when C_H3_REAL_SEL =>
                    s_mac_a_in <= i_harm_array_real(2);
                when C_H4_REAL_SEL =>
                    s_mac_a_in <= i_harm_array_real(3);
                when C_H5_REAL_SEL =>
213                 s_mac_a_in <= i_harm_array_real(4);
                when C_H1_IMAG_SEL =>
                    s_mac_a_in <= i_harm_array_imag(0);
                when C_H2_IMAG_SEL =>
                    s_mac_a_in <= i_harm_array_imag(1);
218             when C_H3_IMAG_SEL =>
                    s_mac_a_in <= i_harm_array_imag(2);
                when C_H4_IMAG_SEL =>
                    s_mac_a_in <= i_harm_array_imag(3);
                when C_H5_IMAG_SEL =>
223                 s_mac_a_in <= i_harm_array_imag(4);
                when C_H1_REAL_CORRECTED_SEL =>
                    s_mac_a_in <= s_h1_real_corrected;
                when C_H1_IMAG_CORRECTED_SEL =>
                    s_mac_a_in <= s_h1_imag_corrected;
228             when C_DC_SEL =>
                    s_mac_a_in <= i_dc_samples;
                when C_H1_REAL_HDI_SEL =>
                    s_mac_a_in <= s_h1_real_hdi;
                when C_H1_IMAG_HDI_SEL =>
233                 s_mac_a_in <= s_h1_imag_hdi;
                when others =>
                    s_mac_a_in <= (others => '-');
            end case;
        end process;
238
        -- muxes c1; c2; c3; c4
        mac_b_mux_p : process(s_mux_b_select)
        begin
                case s_mux_b_select is
243             when "00" => -- +1/3 (p1_real)
                    s_mac_b_in <= ONE_THIRD;
```

```
                when "01" => -- -1/3 (p1_imag)
                    s_mac_b_in <= MINUS_ONE_THIRD;
                when "10" => -- -1/5 (p1)
                    s_mac_b_in <= MINUS_ONE_FIFTH;
                when "11" => -- +pi/4 (for hdi power)
                    s_mac_b_in <= POWER_SCALE;
                when others =>
                    s_mac_b_in <= (others => '-');
            end case;

        end process;


        -- muxes h1r_corrected, h1i_corrected
        mac_c_mux_p : process(s_mux_c_select, s_h1_real_corrected, s_h1_imag_corrected)
            begin
            case s_mux_c_select is
                when '0' =>
                    s_mac_c_in <= s_h1_real_corrected;
                when '1' =>
                    s_mac_c_in <= s_h1_imag_corrected;
                when others =>
                    s_mac_c_in <= (others => '-');
            end case;
        end process;

        ------------------------------------------------------------------
        -- FSM IMPLEMENTATION
        ------------------------------------------------------------------
        -- state machine registers
        correction_fsm_reg : process(i_clk, i_na_reset)
        begin
            if i_na_reset = '0' then
                s_current_state <= IDLE_S;
            elsif rising_edge(i_clk) then
                s_current_state <= s_next_state;
            end if;

        end process;

        -- state machine comb. logic
        correction_fsm_cmb : process(s_current_state, i_busy, i_ack_data)
        begin
            -- hdi
            s_h1_real_hdi_enable    <= '0';
            s_h1_imag_hdi_enable    <= '0';
            s_p1_real_hdi_enable    <= '0';
            s_p1_imag_hdi_enable    <= '0';
            s_dc_squared_enable     <= '0';
            s_ms_corrected_enable   <= '0';
            --hd5
            s_hd5_numerator_enable  <= '0';
            s_hd5_denumerator_enable<= '0';
            -- mac enable squaring mode
            s_en_square <= '0';

            s_mux_a_select <= (others => '0');
            s_mux_b_select <= (others => '0');
            s_mux_c_select <= '0';

            s_h1_real_corrected_enable <= '0';
            s_h1_imag_corrected_enable <= '0';
            s_h1_load <= '0';

            s_clear_regs <= '0';

            o_ack_data <= '0';
            o_done <= '0';

            s_next_state <= s_current_state;

            case s_current_state is
                -- IDLE
                when IDLE_S =>
                    s_clear_regs <= '1'; -- clear all registers
                    if i_busy = '0' then
                        s_next_state <= INIT_REGS_S;
                    end if;

                -- LOAD REGISTERS AND DO THE MS CORRECTION
                when INIT_REGS_S =>
                    s_h1_load <= '1';
                    s_next_state <= REAL_CORRECTION_1_S;
                    -- MS * N
                    s_ms_corrected_enable <= '1';
```

```
                -- RAFT CORRECTION START
                when REAL_CORRECTION_1_S => -- h1+=1/3*h3
333                 s_mux_a_select <= C_H3_REAL_SEL;    -- h3
                    s_mux_b_select <= "00";       -- +1/3
                    s_mux_c_select <= '0';        -- write back to h1
                    s_h1_real_corrected_enable <= '1';
                    s_next_state <= REAL_CORRECTION_2_S;
338
                when REAL_CORRECTION_2_S => -- h1-=1/5*h5
                    s_mux_a_select <= C_H5_REAL_SEL;    -- h5
                    s_mux_b_select <= "10";       -- -1/5
                    s_mux_c_select <= '0';        -- write back to h1
343                 s_h1_real_corrected_enable <= '1';
                    s_next_state <= IMAG_CORRECTION_1_S;

                when IMAG_CORRECTION_1_S => -- h1-=1/3*h3
                    s_mux_a_select <= C_H3_IMAG_SEL;    -- h3
348                 s_mux_b_select <= "01";       -- -1/3
                    s_mux_c_select <= '1';        -- write back to h1
                    s_h1_imag_corrected_enable <= '1';
                    s_next_state <= IMAG_CORRECTION_2_S;

353             when IMAG_CORRECTION_2_S => -- h1-=1/5*h5
                    s_mux_a_select <= C_H5_IMAG_SEL;    -- h5
                    s_mux_b_select <= "10";       -- -1/5
                    s_mux_c_select <= '1';        -- write back to h1
                    s_h1_imag_corrected_enable <= '1';
358                 s_next_state <= POWER_CORRECTION_REAL_S;
                -- RAFT CORRECTION END

                -- HDI PRECALCULATIONS / CORRECTIONS START
                when POWER_CORRECTION_REAL_S =>
363                 if c_en_one_period = '1' then
                        s_mux_a_select <= C_H1_REAL_SEL; -- h1_real
                    else
                        s_mux_a_select <= C_H2_REAL_SEL; -- h2_real
                    end if;
368                 s_mux_b_select <= "11";       -- powerscale
                    s_h1_real_hdi_enable <= '1';
                    s_next_state <= POWER_CORRECTION_IMAG_S;

                when POWER_CORRECTION_IMAG_S =>
373                  if c_en_one_period = '1' then
                        s_mux_a_select <= C_H1_IMAG_SEL; -- h1_imag
                    else
                        s_mux_a_select <= C_H2_IMAG_SEL; -- h2_imag
                    end if;
378                 s_mux_b_select <= "11";       -- powerscale
                    s_h1_imag_hdi_enable <= '1';
                    s_next_state <= SQUARE_DC_S;

                when SQUARE_DC_S =>
383                 s_mux_a_select <= C_DC_SEL;
                    s_en_square <= '1';
                    s_dc_squared_enable <= '1';
                    s_next_state <= SQUARE_H1_REAL_HDI_S;

388             when SQUARE_H1_REAL_HDI_S =>
                    s_mux_a_select <= C_H1_REAL_HDI_SEL;
                    s_en_square <= '1';
                    s_p1_real_hdi_enable <= '1';
                    s_next_state <= SQUARE_H1_IMAG_HDI_S;
393
                when SQUARE_H1_IMAG_HDI_S =>
                    s_mux_a_select <= C_H1_IMAG_HDI_SEL;
                    s_en_square <= '1';
                    s_p1_imag_hdi_enable <= '1';
398                 s_next_state <= ADD_P5_REAL_S;
                -- HDI PRECALCULATIONS / CORRECTIONS END

                -- HD5 CALCULATIONS OF NUM AND DENUM START
                when ADD_P5_REAL_S =>
403                 s_mux_a_select <= C_H5_REAL_SEL;
                    s_hd5_denumerator_enable <= '1';
                    s_en_square <= '1';
                    s_next_state <= ADD_P5_IMAG_S;

408             when ADD_P5_IMAG_S =>
                    s_mux_a_select <= C_H5_IMAG_SEL;
                    s_hd5_denumerator_enable <= '1';
                    s_en_square <= '1';
                    s_next_state <= ADD_P4_REAL_S;
413
                when ADD_P4_REAL_S =>
```

```
                    s_mux_a_select <= C_H4_REAL_SEL;
                    s_hd5_denumerator_enable <= '1';
                    s_en_square <= '1';
                    s_next_state <= ADD_P4_IMAG_S;

                when ADD_P4_IMAG_S =>
                    s_mux_a_select <= C_H4_IMAG_SEL;
                    s_hd5_denumerator_enable <= '1';
                    s_en_square <= '1';
                    s_next_state <= ADD_P3_REAL_S;

                when ADD_P3_REAL_S =>
                    s_mux_a_select <= C_H3_REAL_SEL;
                    s_hd5_denumerator_enable <= '1';
                    s_en_square <= '1';
                    s_next_state <= ADD_P3_IMAG_S;

                when ADD_P3_IMAG_S =>
                    s_mux_a_select <= C_H3_IMAG_SEL;
                    s_hd5_denumerator_enable <= '1';
                    s_en_square <= '1';
                    if c_en_one_period = '1' then
                        s_next_state <= ADD_P2_REAL_S;
                    else
                        s_next_state <= ADD_P1_REAL_S;
                    end if;

                when ADD_P2_REAL_S =>
                    s_mux_a_select <= C_H2_REAL_SEL;
                    s_hd5_denumerator_enable <= '1';
                    s_en_square <= '1';
                    s_next_state <= ADD_P2_IMAG_S;
                    if c_en_one_period <= '0' then
                        s_hd5_numerator_enable <= '1';
                    end if;

                when ADD_P2_IMAG_S =>
                    s_mux_a_select <= C_H2_IMAG_SEL;
                    s_hd5_denumerator_enable <= '1';
                    s_en_square <= '1';
                    if c_en_one_period = '1' then
                        s_next_state <= ADD_P1_REAL_S;
                    else
                        s_next_state <= HANDSHAKE_S;
                    end if;

                when ADD_P1_REAL_S =>
                    s_mux_a_select <= C_H1_REAL_SEL;
                    s_hd5_denumerator_enable <= '1';
                    s_en_square <= '1';
                    s_next_state <= ADD_P1_IMAG_S;
                    if c_en_one_period <= '1' then
                        s_hd5_numerator_enable <= '1';
                    end if;

                when ADD_P1_IMAG_S =>
                    s_mux_a_select <= C_H1_IMAG_SEL;
                    s_hd5_denumerator_enable <= '1';
                    s_en_square <= '1';
                    if c_en_one_period = '1' then
                        s_next_state <= HANDSHAKE_S;
                    else
                        s_next_state <= ADD_P2_REAL_S;
                    end if;
                -- HD5 CALCULATIONS OF NUM AND DENUM END

                -- NEXT STAGE HANDSHAKE
                WHEN HANDSHAKE_S =>
                    o_done <= '1'; -- set the signal that we are ready
                    if i_ack_data = '1' then
                        s_next_state <= DONE_S;
                    end if;

                -- PREVIOUS STAGE ACK -> BACK INTO IDLE
                when DONE_S =>
                    o_ack_data <= '1'; -- raise this for one clock cylce
                    s_next_state <= IDLE_S;

                when others =>
                    s_next_state <= IDLE_S;
            end case ;

        end process;
        ------------------------------------------------------------------
```

```
    -- MAC INSTANTIATION
    ------------------------------------------------------------------------
    Inst_correction_mac: correction_mac PORT MAP(
        in_a => s_mac_a_in,
        in_b => s_mac_b_in,
        in_c => s_mac_c_in,
        i_en_square => s_en_square,
        in_ms_correction => i_ms_samples,
        o_a_times_b => s_a_times_b,
        o_ms_times_240 => s_ms_times_240,
        o_a_times_b_plus_c => s_a_times_b_plus_c
    );
    ------------------------------------------------------------------------
    -- HARMONICS ADDER (FOR HD5) (40 bit unsigned adder)
    ------------------------------------------------------------------------
    s_power_output  <= unsigned(std_logic_vector(s_a_times_b));
    s_harm_adder    <= resize(s_power_output, s_harm_adder'length) + s_hd5_denumerator;

    ------------------------------------------------------------------------
    -- REGISTER IMPLEMENTATIONS
    ------------------------------------------------------------------------
    -- h1 correction registers
    h1_corrected_p : process( i_clk, s_reg_reset )
        begin
            if s_reg_reset = '0' then
                s_h1_real_corrected <= (others => '0');
                s_h1_imag_corrected <= (others => '0');
            elsif rising_edge(i_clk) then
                if s_h1_load = '1' then

                    s_h1_real_corrected <= i_harm_array_real(0);
                    s_h1_imag_corrected <= i_harm_array_imag(0);
                else

                    if s_h1_real_corrected_enable = '1' then
                        s_h1_real_corrected <= s_a_times_b_plus_c;
                    end if;

                    if s_h1_imag_corrected_enable = '1' then
                        s_h1_imag_corrected <= s_a_times_b_plus_c;
                    end if;
                end if;
            end if;
    end process;

    -- hdi registers: h1, p1, pms, pdc
    hdi_regs_p : process( i_clk, s_reg_reset )
    begin
        if s_reg_reset = '0' then
            s_ms_corrected  <= (others => '0');
            s_dc_squared    <= (others => '0');
            s_h1_real_hdi   <= (others => '0');
            s_h1_imag_hdi   <= (others => '0');
            s_p1_real_hdi   <= (others => '0');
            s_p1_imag_hdi   <= (others => '0');

        elsif rising_edge(i_clk) then

            if s_ms_corrected_enable = '1' then
                s_ms_corrected <= s_ms_times_240;
            end if;

            if s_dc_squared_enable = '1' then
                s_dc_squared <= unsigned(std_logic_vector(s_a_times_b));
            end if;

            if s_h1_real_hdi_enable = '1' then
                --s_h1_real_hdi <= s_a_times_b(35 downto 18); -- corrected h1 for hdi (h1 * powerscale)
                -- shift only by 9 instead of 18 (10 bit quantisation)
                s_h1_real_hdi <= s_a_times_b(26 downto 9); -- corrected h1 for hdi (h1 * powerscale) / 512
            end if;

            if s_h1_imag_hdi_enable = '1' then
                --s_h1_imag_hdi <= s_a_times_b(35 downto 18); -- corrected h1 for hdi (h1 * powerscale)
                -- shift only by 9 instead of 18 (10 bit quantisation)
                s_h1_imag_hdi <= s_a_times_b(26 downto 9); -- corrected h1 for hdi (h1 * powerscale) / 512
            end if;

            if s_p1_real_hdi_enable = '1' then
                s_p1_real_hdi <= unsigned(std_logic_vector(s_a_times_b));
            end if;

            if s_p1_imag_hdi_enable = '1' then
                s_p1_imag_hdi <= unsigned(std_logic_vector(s_a_times_b));
            end if;
```

```
            end if ;
        end process ; -- ms_dc_p

        -- hd5 registers : numerator and denumerator
        hd5_reg_p : process( i_clk, s_reg_reset )
        begin
            if s_reg_reset = '0' then
                s_hd5_numerator <= (others => '0');
                s_hd5_denumerator <= (others => '0');
            elsif rising_edge(i_clk) then

                if s_hd5_numerator_enable = '1' then
                    s_hd5_numerator <= s_hd5_denumerator;
                end if;

                if s_hd5_denumerator_enable = '1' then
                    s_hd5_denumerator <= s_harm_adder;
                end if;
            end if;

        end process ; -- hd5_reg_p

        ------------------------------------------------------------------
        -- OUTPUT ASSIGNMENTS
        ------------------------------------------------------------------
        -- hdi
        o_p1_real_hdi    <= s_p1_real_hdi;
        o_p1_imag_hdi    <= s_p1_imag_hdi;
        o_dc_power       <= s_dc_squared;
        o_ms_power       <= s_ms_corrected;
        -- hd5
        o_hd5_numerator      <= s_hd5_numerator;
        o_hd5_denumerator    <= s_hd5_denumerator;

        -- debug
        -- hdi
        o_h1_real_hdi        <= s_h1_real_hdi;
        o_h1_imag_hdi        <= s_h1_imag_hdi;
        -- hd5
        o_h1_real_corrected <= s_h1_real_corrected;
        o_h1_imag_corrected <= s_h1_imag_corrected;

    end architecture;
    ------------------------------------------------------------------
    -- ARCHITECTURE END  |  ARCHITECTURE END  |  ARCHITECTURE END  |
    ------------------------------------------------------------------
```

Listing C.10: raft_correction.vhd - Das RAFT-Postprocessing (Korrektur und Vorberechnungen für die Verzerrungsmaße)

```
-- ----------------------------------------------------------------------------
-- Entity: raft_correction_tb
-- ----------------------------------------------------------------------------
-- Copyright 2012
-- Filename              : raft_correction_tb.vhd
-- Creation date         : 2012-12-15
-- Author(s)             : Fabian Zahn
-- Version               : 1.00
-- Description           : Testbench
-- ----------------------------------------------------------------------------
-- File History:
-- Date            Version  Author   Comment
-- ----------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
use work.raft_pkg.all;

ENTITY raft_correction_tb IS
END raft_correction_tb;

ARCHITECTURE behavior OF raft_correction_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT raft_correction
    PORT(
            i_clk : IN  std_logic;
            i_na_reset : IN  std_logic;
            i_busy : IN  std_logic;
            i_harm_array_real : IN  HARM_ARRAY_T;
            i_harm_array_imag : IN  HARM_ARRAY_T;
            i_ms_samples : IN  unsigned(25 downto 0);
            i_dc_samples : IN  signed(17 downto 0);
            i_ack_data : in std_logic;
            o_done : out std_logic;
            o_h1_real_corrected : OUT  signed(17 downto 0);
            o_h1_imag_corrected : OUT  signed(17 downto 0);
            o_ack_data : OUT  std_logic;
            o_ms_power : OUT  unsigned(35 downto 0);
            o_dc_power : OUT  unsigned(35 downto 0);
            o_h1_real_hdi : OUT  signed(17 downto 0);
            o_h1_imag_hdi : OUT  signed(17 downto 0);
            o_p1_real_hdi : OUT  unsigned(35 downto 0);
            o_p1_imag_hdi : OUT  unsigned(35 downto 0);
            o_hd5_numerator : OUT  unsigned(39 downto 0);
            o_hd5_denumerator : OUT  unsigned(39 downto 0)
           );
    END COMPONENT;


   --Inputs
    signal i_clk : std_logic := '0';
    signal i_na_reset : std_logic := '0';
    signal i_busy : std_logic := '0';
    signal i_harm_array_real : HARM_ARRAY_T;
    signal i_harm_array_imag : HARM_ARRAY_T;
    signal i_ms_samples : unsigned(25 downto 0) := (others => '0');
    signal i_dc_samples : signed(17 downto 0) := (others => '0');
    signal i_ack_data : std_logic := '0';

     --Outputs
    signal o_done : std_logic;
    signal o_h1_real_corrected : signed(17 downto 0);
    signal o_h1_imag_corrected : signed(17 downto 0);
    signal o_ack_data : std_logic;
    signal o_ms_power : unsigned(35 downto 0);
    signal o_dc_power : unsigned(35 downto 0);
    signal o_h1_real_hdi : signed(17 downto 0);
    signal o_h1_imag_hdi : signed(17 downto 0);
    signal o_p1_real_hdi : unsigned(35 downto 0);
    signal o_p1_imag_hdi : unsigned(35 downto 0);
    signal o_hd5_numerator : unsigned(39 downto 0);
    signal o_hd5_denumerator : unsigned(39 downto 0);

   -- Clock period definitions
    constant i_clk_period : time := 10 ns;

BEGIN

     -- Instantiate the Unit Under Test (UUT)
    uut: raft_correction PORT MAP (
            i_clk => i_clk,
            i_na_reset => i_na_reset,
            i_busy => i_busy,
```

```
                i_harm_array_real => i_harm_array_real,
                i_harm_array_imag => i_harm_array_imag,
                i_ms_samples => i_ms_samples,
 89             i_dc_samples => i_dc_samples,
                i_ack_data => i_ack_data,
                o_done => o_done,
                o_h1_real_corrected => o_h1_real_corrected,
                o_h1_imag_corrected => o_h1_imag_corrected,
 94             o_ack_data => o_ack_data,
                o_ms_power => o_ms_power,
                o_dc_power => o_dc_power,
                o_h1_real_hdi => o_h1_real_hdi,
                o_h1_imag_hdi => o_h1_imag_hdi,
 99             o_p1_real_hdi => o_p1_real_hdi,
                o_p1_imag_hdi => o_p1_imag_hdi,
                o_hd5_numerator => o_hd5_numerator,
                o_hd5_denumerator => o_hd5_denumerator
              );
104
        -- Clock process definitions
        i_clk_process :process
        begin
             i_clk <= '0';
109          wait for i_clk_period/2;
             i_clk <= '1';
             wait for i_clk_period/2;
        end process;

114
        -- Stimulus process
        stim_proc: process
        begin
             -- hold reset state for 100 ns.
119          i_harm_array_real(0) <= to_signed(0, i_harm_array_real(0)'length);
             i_harm_array_real(1) <= to_signed(0, i_harm_array_real(1)'length);
             i_harm_array_real(2) <= to_signed(0, i_harm_array_real(2)'length);
             i_harm_array_real(3) <= to_signed(0, i_harm_array_real(3)'length);
             i_harm_array_real(4) <= to_signed(0, i_harm_array_real(4)'length);
124
             i_harm_array_imag(0) <= to_signed(0, i_harm_array_imag(1)'length);
             i_harm_array_imag(1) <= to_signed(0, i_harm_array_imag(2)'length);
             i_harm_array_imag(2) <= to_signed(0, i_harm_array_imag(3)'length);
             i_harm_array_imag(3) <= to_signed(0, i_harm_array_imag(4)'length);
129          i_harm_array_imag(4) <= to_signed(0, i_harm_array_imag(5)'length);

             i_busy <= '1'; -- previous stage is busy

             -- STIMULI --
134          wait for 100 ns;
             i_na_reset <= '1'; -- disable reset

             i_ms_samples <= to_unsigned(2**25-1, i_ms_samples'length);
             i_dc_samples <= to_signed(((2**17)-1), i_dc_samples'length);
139
             i_harm_array_real(0) <= to_signed(512, i_harm_array_real(0)'length);
             i_harm_array_real(1) <= to_signed((2**17)-1, i_harm_array_real(1)'length);
             i_harm_array_real(2) <= to_signed(512, i_harm_array_real(2)'length);
             i_harm_array_real(3) <= to_signed(512, i_harm_array_real(3)'length);
144          i_harm_array_real(4) <= to_signed(512, i_harm_array_real(4)'length);

             i_harm_array_imag(0) <= to_signed(512, i_harm_array_imag(1)'length);
             i_harm_array_imag(1) <= to_signed(512, i_harm_array_imag(2)'length);
             i_harm_array_imag(2) <= to_signed(512, i_harm_array_imag(3)'length);
149          i_harm_array_imag(3) <= to_signed(512, i_harm_array_imag(4)'length);
             i_harm_array_imag(4) <= to_signed(512, i_harm_array_imag(5)'length);
             wait for 10 ns;
             i_busy <= '0'; -- start the conversion

154          wait until o_done = '1';
             wait for 1 ns;
             assert o_ms_power = "00011101111111111111111111100010000" report "MS MISMATCH" severity ERROR; -- ...
                  (240*(2**15-1))
             assert o_dc_power = "00111111111111111100000000000000001" report "DC MISMATCH" severity ERROR; -- ...
                  ((2**17)-1)**2

159          wait for 20 ns;
             --i_ack_data <= '1';
             -- new stimuli here

             wait;
164     end process;
    end;
```

Listing C.11: raft_correction_tb.vhd - Testbench des RAFT-Postprocessings

C.8. Postprocessing MAC-Einheit

```
----------------------------------------------------------------------------
-- Entity: correction_mac
----------------------------------------------------------------------------
-- Copyright 2012
-- Filename            : correction_mac.vhd
-- Creation date       : 2012-12-12
-- Author(s)           : Fabian Zahn
-- Version             : 1.00
-- Description         : mac used for the correction of the raft harmonics
--                       this mac is designed to be used as a general purpose
--                       18x18 bit multiplier all other outputs are application
--                       specific and shouldn't be used for gp calculations
----------------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
----------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.raft_pkg.all;

entity correction_mac is
    Port ( in_a                 : in signed (17 downto 0);
           in_b                 : in signed (17 downto 0);
           in_c                 : in signed (17 downto 0);
           i_en_square          : in std_logic;
           in_ms_correction     : in unsigned (25 downto 0);
           o_a_times_b          : out signed (35 downto 0);
           o_ms_times_240       : out unsigned (35 downto 0);
           o_a_times_b_plus_c   : out signed (17 downto 0));
end correction_mac;

architecture Behavioral of correction_mac is

    signal s_product    : signed (35 downto 0);
    signal s_ms_times_15 : unsigned (29 downto 0);
begin
    ----------------------
    -- 18x18 bit multiplier
    s_product           <= in_a * in_a when i_en_square = '1' else
                           in_a * in_b;
    o_a_times_b         <= s_product;

    ----------------------
    -- (18x18 bit) >> 9 + C
    o_a_times_b_plus_c  <= s_product(26 downto 9) + in_c;

    ---------------
    -- 240 = 15 * 16
    --> mul by 15
    s_ms_times_15       <= in_ms_correction * to_unsigned(15,4);
    --> mul by 16 (shifting by 4)
    o_ms_times_240      <= resize((s_ms_times_15 & "0000"), o_ms_times_240'length);

end Behavioral;
```

Listing C.12: correction_mac.vhd - Die für die Korrektur notwendige 18x18 Bit MAC-Einheit

```
-- ------------------------------------------------------------------
-- entity: correction_mac
-- ------------------------------------------------------------------
-- Copyright 2012
-- Filename            : correction_mac.vhd
-- Creation date       : 2012-12-12
-- Author(s)           : Fabian Zahn
-- Version             : 1.00
-- Description         : correction mac testbench
-- ------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
-- ------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity correction_mac_tb is
end correction_mac_tb;

architecture behavior of correction_mac_tb is
    -- Component Declaration for the Unit Under Test (UUT)
    component correction_mac
    port(
         in_a : IN  signed(17 downto 0);
         in_b : IN  signed(17 downto 0);
         in_c : IN  signed(17 downto 0);
         i_en_square : IN  std_logic;
         in_ms_correction : IN  unsigned(25 downto 0);
         o_a_times_b : OUT  signed(35 downto 0);
         o_ms_times_240 : OUT  unsigned(35 downto 0);
         o_a_times_b_plus_c : OUT  signed(17 downto 0)
        );
    end component;
   --Inputs
   signal in_a : signed(17 downto 0) := (others => '0');
   signal in_b : signed(17 downto 0) := (others => '0');
   signal in_c : signed(17 downto 0) := (others => '0');
   signal i_en_square : std_logic := '0';
   signal in_ms_correction : unsigned(25 downto 0) := (others => '0');
    --Outputs
   signal o_a_times_b           : signed(35 downto 0);
   signal o_ms_times_240        : unsigned(35 downto 0);
   signal o_a_times_b_plus_c    : signed(17 downto 0);
begin
    -- Instantiate the Unit Under Test (UUT)
   uut: correction_mac PORT MAP (
          in_a => in_a,
          in_b => in_b,
          in_c => in_c,
          i_en_square => i_en_square,
          in_ms_correction => in_ms_correction,
          o_a_times_b => o_a_times_b,
          o_ms_times_240 => o_ms_times_240,
          o_a_times_b_plus_c => o_a_times_b_plus_c
        );
   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
        in_a <= to_signed(512, in_a'length);
        in_b <= to_signed(512, in_b'length);
        in_c <= to_signed(1000, in_c'length);
        in_ms_correction <= to_unsigned((511*511*240), in_ms_correction'length);
        wait for 1 ns;
        assert o_a_times_b = to_signed(262144, o_a_times_b'length) report "ERROR -> 512*512" severity ERROR;
        assert o_a_times_b_plus_c = to_signed(1512, o_a_times_b_plus_c'length) report "ERROR -> 512*512 / 512 ...
            + 1000" severity ERROR;

        wait for 100 ns;
        in_a <= to_signed(170, in_a'length);
        in_b <= to_signed(-3066, in_b'length);
        in_c <= to_signed(0, in_c'length);
        wait for 1 ns;
        assert o_a_times_b_plus_c = to_signed(-1019, o_a_times_b_plus_c'length) report "ERROR -> ...
            170*-3066/512" severity ERROR;
      wait;
   end process;
end;
```

Listing C.13: correction_mac_tb.vhd - Testbench der 18x18 Bit MAC-Einheit

C.9. RAFT Package

```
2   -- Package: raft_pkg
    -- Copyright 2012
    -- Filename            : raft_pkg.vhd
    -- Creation date       : 2012-12-15
7   -- Author(s)           : Fabian Zahn
    -- Version             : 1.00
    -- Description         : type definitions
    -- File History:
12  -- Date          Version   Author    Comment

    library ieee;
    use ieee.std_logic_1164.all;
17  use ieee.numeric_std.all;

    package raft_pkg is
        type HARM_ARRAY_T is array(0 to 4) of signed(17 downto 0);
        type HARM_ARRAY_SQUARED_T is array(0 to 4) of signed(35 downto 0);
22  end raft_pkg;

    package body raft_pkg is

    end raft_pkg;
```

Listing C.14: raft_pkg.vhd - Typendefinitionen für das RAFT-Verfahren

D. VHDL-Implementierung der aufwandsreduzierten Interpolation

D.1. Toplevel des Interpolators

```
--------------------------------------------------------------------------------
-- Entity: interpolate_toplevel
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename            : interpolate_toplevel.vhd
-- Creation date       : 2012-11-08
-- Author(s)           : Fabian Zahn
-- Version             : 1.00
-- Description         : 2 stage interpolator toplevel
--                       (interpolate by 4)
--------------------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;

entity interpolate_toplevel is
    Port ( i_clk : in  std_logic;
           i_n_reset_a : in std_logic;
           i_ack_sample : in  std_logic;
           i_start : in  std_logic;
           i_data : in  signed (9 downto 0);
           o_data : out  signed (9 downto 0);
           o_busy : out  std_logic;
           o_sample_valid : out std_logic
           );
end interpolate_toplevel;


architecture Behavioral of interpolate_toplevel is

    COMPONENT interpolate_and_filter
    PORT(
        i_clk : in std_logic;
        i_n_reset_a : in std_logic;
        i_start : in std_logic;
        i_ack_sample : in std_logic;
        i_data : in signed(9 downto 0);
        o_data : out signed(9 downto 0);
        o_sample_valid : out std_logic;
        o_busy : out std_logic
        );
    END COMPONENT;

    signal stage1_output_data : signed(9 downto 0);
    signal stage1_sample_valid : std_logic;
    signal stage1_busy : std_logic;
    signal stage2_busy : std_logic;
    signal stage2_start : std_logic;
    signal stage1_ack_sample : std_logic;

begin

    stage1_i: interpolate_and_filter PORT MAP(
        i_clk => i_clk,
```

```
            i_n_reset_a => i_n_reset_a,
59          i_start => i_start,
            i_ack_sample => stage1_ack_sample,
            i_data => i_data,
            o_data => stage1_output_data,
            o_sample_valid => stage1_sample_valid,
64          o_busy => stage1_busy
        );

        stage2_i: interpolate_and_filter PORT MAP(
            i_clk => i_clk,
69          i_n_reset_a => i_n_reset_a,

            --i_start => stage2_start_sync,
            i_start => stage2_start,

74          i_ack_sample => i_ack_sample,

            --i_data => stage1_output_data_sync,
            i_data => stage1_output_data,
            o_data => o_data,
79          o_sample_valid => o_sample_valid,
            o_busy => stage2_busy
        );

        stage2_start <= stage1_sample_valid and stage1_busy;
84      o_busy <= stage1_busy or stage2_busy;
        stage1_ack_sample <= not stage2_busy;

    end Behavioral;
```

Listing D.1: interpolate_toplevel.vhd - Toplevel des aufwandsreduzierten Interpolators (Interpolation um den Faktor 4)

```
--------------------------------------------------------------------------------
-- Entity: interpolate_toplevel_tb
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename            : interpolate_toplevel_tb.vhd
-- Creation date       : 2012-11-08
-- Author(s)           : Fabian Zahn
-- Version             : 1.00
-- Description         : testbench
--------------------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;

ENTITY interpolate_toplevel_tb IS
END interpolate_toplevel_tb;

ARCHITECTURE behavior OF interpolate_toplevel_tb IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT interpolate_toplevel
    PORT(
         i_clk : IN  std_logic;
         i_n_reset_a : IN  std_logic;
         i_ack_sample : IN  std_logic;
         i_start : IN  std_logic;
         i_data : IN  signed(9 downto 0);
         o_data : OUT  signed(9 downto 0);
         o_busy : OUT  std_logic;
         o_sample_valid : OUT  std_logic
        );
    END COMPONENT;

   --Inputs
   signal i_clk : std_logic                := '0';
   signal i_n_reset_a : std_logic          := '0';
   signal i_ack_sample : std_logic         := '0';
   signal i_start : std_logic              := '0';
   signal i_data : signed(9 downto 0)      := (others => '0');

    --Outputs
   signal o_data : signed(9 downto 0);
   signal o_busy : std_logic;
   signal o_sample_valid : std_logic;

   -- Clock period definitions
   constant i_clk_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: interpolate_toplevel PORT MAP (
          i_clk => i_clk,
          i_n_reset_a => i_n_reset_a,
          i_ack_sample => i_ack_sample,
          i_start => i_start,
          i_data => i_data,
          o_data => o_data,
          o_busy => o_busy,
          o_sample_valid => o_sample_valid
        );

   -- Clock process definitions
   i_clk_process :process
   begin
        i_clk <= '0';
        wait for i_clk_period/2;
        i_clk <= '1';
        wait for i_clk_period/2;
   end process;

   -- Stimulus process
   stim_proc: process
   begin
        i_n_reset_a <= '0';
      wait for 25 ns;
        -- disable reset
        i_n_reset_a <= '1';
        ------------------------
        -- process 1st sample
        i_data <= to_signed(511, i_data'length);
        i_start <= '1';
```

```
            -- wait for the fsm to start
            wait until o_busy = '1';
 88         i_start <= '0';
            -- get one sample
            wait until o_sample_valid = '1';
            wait for 10 ns;
            i_ack_sample <= '1';
 93         wait until o_sample_valid = '0';
            wait for 10 ns;
            i_ack_sample <= '0';
            -- get the 2nd sample
            wait until o_sample_valid = '1';
 98         wait for 10 ns;
            i_ack_sample <= '1';
            wait until o_sample_valid = '0';
            wait for 10 ns;
            i_ack_sample <= '0';
103
            wait until o_sample_valid = '1';
            wait for 10 ns;
            i_ack_sample <= '1';
            wait until o_sample_valid = '0';
108         wait for 10 ns;
            i_ack_sample <= '0';
            -- get the 2nd sample
            wait until o_sample_valid = '1';
            wait for 10 ns;
113         i_ack_sample <= '1';
            wait until o_sample_valid = '0';
            wait for 10 ns;
            i_ack_sample <= '0';
            ------------------------------
118         -- process 2nd sample
            i_data <= to_signed(0, i_data'length);
            i_start <= '1';
            wait until o_busy = '1';
            i_start <= '0';
123         -- get one sample
            wait until o_sample_valid = '1';
            wait for 10 ns;
            i_ack_sample <= '1';
            wait until o_sample_valid = '0';
128         wait for 10 ns;
            i_ack_sample <= '0';
            -- get the 2nd sample
            wait until o_sample_valid = '1';
            wait for 10 ns;
133         i_ack_sample <= '1';
            wait until o_sample_valid = '0';
            wait for 10 ns;
            i_ack_sample <= '0';

138         wait until o_sample_valid = '1';
            wait for 10 ns;
            i_ack_sample <= '1';
            wait until o_sample_valid = '0';
            wait for 10 ns;
143         i_ack_sample <= '0';
            -- get the 2nd sample
            wait until o_sample_valid = '1';
            wait for 10 ns;
            i_ack_sample <= '1';
148         wait until o_sample_valid = '0';
            wait for 10 ns;
            i_ack_sample <= '0';
            ------------------------------
            -- process 3rd sample
153         i_data <= to_signed(0, i_data'length);
            i_start <= '1';
            wait until o_busy = '1';
            i_start <= '0';
            -- get one sample
158         wait until o_sample_valid = '1';
            wait for 10 ns;
            i_ack_sample <= '1';
            wait until o_sample_valid = '0';
            wait for 10 ns;
163         i_ack_sample <= '0';
            -- get the 2nd sample
            wait until o_sample_valid = '1';
            wait for 10 ns;
            i_ack_sample <= '1';
168         wait until o_sample_valid = '0';
            wait for 10 ns;
            i_ack_sample <= '0';
```

```
            wait until o_sample_valid = '1';
            wait for 10 ns;
173         i_ack_sample <= '1';
            wait until o_sample_valid = '0';
            wait for 10 ns;
            i_ack_sample <= '0';
            -- get the 2nd sample
178         wait until o_sample_valid = '1';
            wait for 10 ns;
            i_ack_sample <= '1';
            wait until o_sample_valid = '0';
            wait for 10 ns;
183         i_ack_sample <= '0';
          wait;
      end process;
  end;
```

Listing D.2: interpolate_toplevel_tb.vhd - Testbench des Toplevels

D.2. Linearer Interpolator um den Faktor zwei

```
--------------------------------------------------------------------------------
-- Entity: interpolate_and_filter
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename           : interpolate_and_filter.vhd
-- Creation date      : 2012-11-08
-- Author(s)          : Fabian Zahn
-- Version            : 1.00
-- Description        : implementation of an upsampler with
--                      interpolation filter (factor of 2)
--------------------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;

entity interpolate_and_filter is
    Port (  i_clk : in std_logic;
            i_n_reset_a : in std_logic;
            i_start : in std_logic;
            i_ack_sample : in  std_logic;
            i_data : in signed(9 downto 0);
            o_data : out signed(9 downto 0);
            o_sample_valid : out std_logic;
            o_busy : out std_logic
          );
end interpolate_and_filter;

architecture Behavioral of interpolate_and_filter is

-- fsm states
subtype FSM_STATE_T is unsigned(2 downto 0);
constant IDLE_S                 : FSM_STATE_T := "000";
constant PROCESS_SAMPLE_S   : FSM_STATE_T := "001";
constant OUTPUT_SAMPLE_1_S  : FSM_STATE_T := "010";
constant INSERT_ZERO_S      : FSM_STATE_T := "011";
constant OUTPUT_SAMPLE_2_S  : FSM_STATE_T := "100";

-- zero sample value for insertion in the interpolation process
constant ZERO_SAMPLE    : signed(9 downto 0) := (others => '0');
signal current_state, next_state : FSM_STATE_T;
-- misc signals
signal en_shift_registers   : std_logic;
signal sel_mux              : std_logic;
signal z0, z1, z2       : signed(9 downto 0);
signal tmp_result_0 : signed (10 downto 0);
signal tmp_result_1 : signed (9 downto 0);

begin
    -- FSM register
    fsm_reg : process(i_clk, i_n_reset_a)
    begin
        if i_n_reset_a = '0' then
            current_state <= IDLE_S;
        elsif rising_edge(i_clk) then
            current_state <= next_state;
        end if;
    end process;

    -- FSM logic
    fsm_logic : process(current_state, i_start, i_ack_sample)
    begin
        -- default assignments
            -- outputs
        o_sample_valid <= '0';
        o_busy <= '1';
            -- internal signals
        en_shift_registers <= '0';
        sel_mux <= '0';
            -- state type
        next_state <= current_state;

        case current_state is
            -- idle
            when IDLE_S =>
                if i_start = '1' then
                    next_state <= PROCESS_SAMPLE_S;
```

```
                    end if;
                    o_busy <= '0';

                -- process input sample
                when PROCESS_SAMPLE_S =>
                    next_state <= OUTPUT_SAMPLE_1_S;
                    en_shift_registers <= '1';

                -- output filtered value
                when OUTPUT_SAMPLE_1_S =>
                    if i_ack_sample = '1' then
                        next_state <= INSERT_ZERO_S;
                    end if;
                    o_sample_valid <= '1';

                -- inser a zero (interpolation by 2)
                when INSERT_ZERO_S =>
                    next_state <= OUTPUT_SAMPLE_2_S;
                    sel_mux <= '1';
                    en_shift_registers <= '1';

                -- output filtered value no. 2
                when OUTPUT_SAMPLE_2_S =>
                    if i_ack_sample = '1' then
                        next_state <= IDLE_S;
                    end if;
                    o_sample_valid <= '1';

                when others =>
                    next_state <= IDLE_S;
            end case;
        end process;
        -- delay chain register process
        delay_chain : process(i_clk, i_n_reset_a)
        begin
            if i_n_reset_a = '0' then
                z0 <= (others => '0');
                z1 <= (others => '0');
                z2 <= (others => '0');
            elsif rising_edge(i_clk) then
                if en_shift_registers = '1' then
                    z2 <= z1;
                    z1 <= z0;
                    if sel_mux = '0' then
                        z0 <= i_data;
                    else
                        z0 <= ZERO_SAMPLE;
                    end if;
                end if;
            end if;
        end process;
        -- interpolation logic H(z) = (.5 z^0 + z^-1 + .5 z^-2)
        --  0.5 * z0 + 0.5 * z2
        tmp_result_0 <= resize(z0, z0'length+1) + resize(z2, z2'length+1);
        -- (0.5 * z0 + 0.5 * z2) + z1
        tmp_result_1 <= tmp_result_0(10 downto 1) + z1;
        o_data <= tmp_result_1;
    end Behavioral;
```

Listing D.3: interpolate_and_filter.vhd - Linearer Interpolator (um den Faktor 2)

```
--------------------------------------------------------------------------------
-- Entity: interpolate_and_filter_tb
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename            : interpolate_and_filter_tb.vhd
-- Creation date       : 2012-11-08
-- Author(s)           : Fabian Zahn
-- Version             : 1.00
-- Description         : testbench
--------------------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;

ENTITY interpolate_and_filter_tb IS
END interpolate_and_filter_tb;

ARCHITECTURE behavior OF interpolate_and_filter_tb IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT interpolate_and_filter
    PORT(
         i_clk : IN  std_logic;
         i_n_reset_a : IN  std_logic;
         i_start : IN  std_logic;
         i_ack_sample : IN  std_logic;
         i_data : IN  signed(9 downto 0);
         o_data : OUT  signed(9 downto 0);
         o_sample_valid : OUT  std_logic;
         o_busy : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal i_clk : std_logic := '0';
   signal i_n_reset_a : std_logic := '0';
   signal i_start : std_logic := '0';
   signal i_ack_sample : std_logic := '0';
   signal i_data : signed(9 downto 0) := (others => '0');

    --Outputs
   signal o_data : signed(9 downto 0);
   signal o_sample_valid : std_logic;
   signal o_busy : std_logic;

   -- Clock period definitions
   constant i_clk_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: interpolate_and_filter PORT MAP (
          i_clk => i_clk,
          i_n_reset_a => i_n_reset_a,
          i_start => i_start,
          i_ack_sample => i_ack_sample,
          i_data => i_data,
          o_data => o_data,
          o_sample_valid => o_sample_valid,
          o_busy => o_busy
        );

   -- Clock process definitions
   i_clk_process :process
   begin
        i_clk <= '0';
        wait for i_clk_period/2;
        i_clk <= '1';
        wait for i_clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
        i_n_reset_a <= '0';
      wait for 25 ns;
        -- disable reset
        i_n_reset_a <= '1';

        ----------------------
```

```
            -- process 1st sample
            i_data <= to_signed(511, i_data'length);
88          i_start <= '1';

            -- wait for the fsm to start
            wait until o_busy = '1';
            i_start <= '0';
93
            -- get one sample
            wait until o_sample_valid = '1';
            i_ack_sample <= '1';
            wait until o_sample_valid = '0';
98          i_ack_sample <= '0';
            -- get the 2nd sample
            wait until o_sample_valid = '1';
            i_ack_sample <= '1';
            wait until o_sample_valid = '0';
103         i_ack_sample <= '0';


            ----------------------
            -- process 2nd sample
108         i_data <= to_signed(0, i_data'length);
            i_start <= '1';
            wait until o_busy = '1';
            i_start <= '0';

113         -- get one sample
            wait until o_sample_valid = '1';
            i_ack_sample <= '1';
            wait until o_sample_valid = '0';
            i_ack_sample <= '0';
118         -- get the 2nd sample
            wait until o_sample_valid = '1';
            i_ack_sample <= '1';
            wait until o_sample_valid = '0';
            i_ack_sample <= '0';
123


        wait;
    end process;
128
END;
```

Listing D.4: interpolate_and_filter_tb.vhd - Testbench des linearen Interpolators

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 5. April 2013	
Ort, Datum	Unterschrift