Fachhochschule Westküste
Hochschule für Wirtschaft & Technik

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Ada Koundoul

Signalverarbeitung für ein magnetisches Sensor-Array als digitaler Chipentwurf

Fachhochschule Westküste
Fachbereich Technik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

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

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

Ada Koundoul

Signalverarbeitung für ein magnetisches Sensor-Array als digitaler Chipentwurf

Masterarbeit eingereicht im Rahmen der Masterprüfung
im gemeinsamen Studiengang 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.-Ing. Karl Ragmar Riemschneider
Zweitgutachter: Prof. Dr.-Ing. Hans-Dieter Schütte

Abgegeben am 24. Oktober 2018

Ada Koundoul

Thema der Masterarbeit

Signalverarbeitung für ein magnetisches Sensor-Array als digitaler Chipentwurf

Stichworte

Sensor-Array (8×8), Interpolation, zweidimensionale diskrete Fouriertransformation (2D-DFT), 2D-Filter, FPGA-Xilinx-Vivado, Cadence, ASIC

Kurzzusammenfassung

In der vorliegenden Masterarbeit werden für ein 8x8 magnetoresistives Sensor-Array drei Signalverarbeitungsmodule in VHDL entworfen. Das erste Modul, die Interpolation, erweitert die Sensordaten auf eine Größe von 15x15. Die zweidimensionalen diskreten Fouriertransformation, als zweites Modul, transformiert die Daten vom Orts- in den Bildbereich transformiert. Das dritte Modul besteht aus einer Filterung im Bildbereich und dient der Separation von relevanten und irrelevanten Information. Die Module werden zunächst auf einem FPGA getestet. Des Weiteren folgt eine Synthese mit dem Chipdesigntool “Cadence”, um eine Abschätzung des Aufwands auf einem Chip zu geben.

Ada Koundoul

Title of the master thesis

Signal processing for a magnetic sensor Array as a digital chip draft

Keywords

Sensor array (8×8), Interpolation, two dimensional discrete Fourier transform (2D-DFT), 2D filter, FPGA-Xilinx-Vivado, Cadence, ASIC

Abstract

In this master thesis three signal processing modules for a 8x8 magnetoresistive sensor Array are designed in VHDL. The first module, the interpolation, extends the sensor data to a size of 15x15. As the second module the discrete two-dimensional Fourier transform transforms the data from spatial domain to frequency domain. The third module is a two dimensional filter in the frequency domain. This filter is used for the separation of relevant and irrelevant information. The modules are tested on a FPGA. An analysis of the effort on a chip is applied by the synthesis with the chip design tool “Cadence”.

Vorwort

Die Forschungen zur vorliegenden Arbeit wurden in der Zeit von April 2018 bis Oktober 2018 in dem Forschungsverbundprojekt ISAR an der Hochschule für Angewandte Wissenschaften Hamburg durchgeführt.

Ein großer Dank gilt Prof. Dr.-Ing. K.-R. Riemschneider, den ich sehr für den Umgang mit seinen Studenten und Mitarbeitern schätze. Dank der tollen Projektleitung habe ich mich in dem Team sehr wohl und gut aufgehoben gefühlt.

Weiteren Dank spreche ich meinem Zweitprüfer Prof. Dr.-Ing. Hans-Dieter Schütte für die Über- nahme der Prüfung und das Interesse an meinem Thema aus.

Besondere Unterstützung habe ich von M.Sc. T. Schüthe erhalten. Er hat viele Stunden investiert und mir stets mit Rat und Tat zur Seite gestanden. Dafür bin ich ihm von ganzem Herzen dankbar.

Dipl.-Ing. Herrn G. Müller möchte ich hier ebenso erwähnen und für die Zeit und Mühe bei den Korrekturen danken. Die kritischen und ehrlichen Hinweise und Gespräche haben mir sehr geholfen.

Mit sehr viel Freude habe ich in der Projektgruppe gearbeitet und meine Kommilitonen sehr ins Herz geschlossen. Auch ihnen spreche ich meine Dankbarkeit für die tolle Arbeitsatmosphäre und Unterstützung aus.

Unendlichen Dank empfinde ich für meine beiden Onkel Bacar und Baba Gadji. Bacar Gadji gab mir stets halt und hatte immer ein offenes Ohr für mich. Ohne die Beiden wäre vieles nicht möglich gewesen. Unendlichen Dank an meiner Mutter Daga Gadji, sie hat mir stets den Rücken für das Studium freigehalten und mich durch alle Lebenslagen getragen. Liebes Dank an meiner Frau Asta Gadji und besonders Dank an meiner Tante Maimouna Seck und der ganzen Familie aus Lübeck.

Inhaltsverzeichnis

Abbildungsverzeichnis **VIII**

1 Einleitung **1**
1.1 Stand der Technik 1
1.2 Motivation 2
1.3 Ziel dieser Arbeit 2

2 Grundlagen **4**
2.1 Zahlensysteme 4
2.1.1 Natürliche Zahlen 4
2.1.2 Ganze Zahlen (Zweierkomplement-Darstellung) 5
2.1.3 Darstellung reeller Zahlen 5
2.2 Prinzip der Interpolation nach Matlab-interp2() 10
2.3 Diskrete Fouriertransformation (DFT) 11
2.3.1 Definition der Zweidimensionalen Diskreten Fouriertransformation (2D-DFT) 12
2.3.2 Die Matrixrepräsentation der 2D-DFT 12
2.4 Das 2D-Digitalfilter 13
2.4.1 Grundstruktur linearer digitaler Filter 14
2.4.2 Tiefpassfilterung im Frequenzbereich 14

3 Entwicklungskonzept **16**
3.1 Systementwurf auf dem FPGA 17
3.2 Entity des Moduls 18
3.3 Der System-Multiplexer 19
3.4 Speicherverwaltung 20
3.5 Datenformat im Speicher 22
3.6 Speicherzugriff 23
3.6.1 Lesevorgang 23
3.6.2 Schreibvorgang 24
3.7 Das Zahlenformat S1Q10 24
3.8 Hauptfunktionen 25
3.8.1 Ermittlung des Mittelwertes in Hardware 25
3.8.2 Die Addition und Subtraktion zweier Festkommazahlen S1Q10 in Hardware 26
3.8.3 Das Produkt zweier Festkommazahlen S1Q10 in Hardware 27
3.9 Das Entwicklungsverfahren in VHDL 28

4 Interpolation **30**
4.1 Modularchitektur 30
4.1.1 Die Komponente `ROWINT` (horizontale Interpolation) 31
4.1.2 Die Komponente `COLINT` (vertikale Interpolation) 34

5 Die Zweidimensionale diskrete Fouriertransformation (2D-DFT) **37**
5.1 Bestimmung der Twiddle-Faktoren 38
5.2 Modularchitekture 39
5.3 1D-DFT 39
5.3.1 Komponente addUnit 40
5.3.2 Komponente OperatingUnit 42
5.4 2D_DFT 45
5.4.1 Komponente COLDFT 46
5.4.2 Komponente DFT2D 47

6 2D-Digitalfilter **49**
6.0.1 Berechnungskonzept des Filters 50
6.0.2 Implementierung in VHDL 50

7 Evaluation, Darstellung der Ergebnisse **53**
7.1 Evaluation der VHDL Interpolation-Modul 53
7.2 Evaluation der VHDL 2D-DFT-Modul 54
7.2.1 Evaluation der VHDL 1D-MOD für 1D-DFT 54
7.2.2 Evaluation der VHDL 1D-MOD für 2D-DFT 54
7.3 Evaluation der VHDL 2D-Filter 56
7.4 Synthese auf FPGA-Ebene, schematic 57
7.5 Synthese auf FPGA-Ebene, Evaluation der Standardzellen 58
7.6 Synthese auf Cadence 59

8 Schlussfolgerungen **60**
8.1 Zusammenfassung 60
8.2 Ausblick 61

Literatur **64**

Anhang

A Quellcode **66**

Selbstständigkeitserklärung **105**

Abbildungsverzeichnis

1.1 Eine vereinfachte graphische Darstellung eines Signalverarbeitungssystems 1
1.2 Erfassung der Sensorinformation bei einem Permantenmagneten. Quelle: Projekt ISAR – HAW Hamburg 2

2.1 Addition zweier 8 Bit langen Festkommazahlen 7
2.2 Zahlenkreis für 4 Bit Zahlen in Zweierkomplement-Darstellung [3] 8
2.3 Ermittlung von interpolierten Sensordaten mittels Matlab_interp2(). Quelle: BA-Rindelaub HAW-Hamburg 11
2.4 Diskrete Transformation der Signalfolge x(k) in die Spektralfolge X(l) . . 12
2.5 Ablaufplan der Digitalisierung analoger Signale mittels PC oder Digital Signal Processing (DSP) 14
2.6 Signalverlauf mit digitalem Filter 14
2.7 Gauss-Tiefpassfilter nach [6] 15
2.8 Gauss-Tiefpassfilter Bildverarbeitung Prof. Kölzer HAW-Hamburg 15

3.1 Die Testplattform besteht aus dem Mikrocontroller TM4C1294 auf dem Evalutionsboard Connected Lauchpad von Texas Instruments und dem FPGA-Zedboard von Xilinx 16
3.2 Systementwurf auf FPGA 17
3.3 Allgemeine Schnittstellen der am BRAM angeschlossenen Signalverarbeitungsmodule 18
3.4 System-Multiplexer in Modulbeschreibung 20
3.5 Signalflussplan der Verarbeitung der Sensordaten 21
3.6 Speicherverwaltung und Datenaufteilung im BRAM Speicher 22
3.7 Darstellung der Sensordaten in einer 32 Bit BRAM-Speicherzelle 23
3.8 Lesevorgang beim Speicherzugriff 23
3.9 Schreibvorgang beim Speicherzugriff 24
3.10 Das 12 Bit-Festkommazahlenformat S1Q10 25
3.11 Berechnung des Mittelwertes zweier 12 Bit Festkommazahlen 26
3.12 Addition zweier Dualzahlen in S1Q10-Format 27
3.13 Resultat des Produktes zweier Dualzahlen in S1Q10-Format 28
3.14 Aufteilung der Signalverarbeitung anhand Dreiprozessdarstellung 28

4.1 Interpolationstest aus (8×8) zu (15×15) mittels Matlab interp2() . . . 30
4.2 Architektur des Moduls Interpolation 31
4.3 Komplexe Sensordaten (Eingangsmatrix (8×8)) 31

4.4 Auswahl der Ausgangsschnittstellen anhand eines internen MUX. Hiermit werden Signalkonflikte vermieden 32
4.5 Datenreihenfolge im BRAM-Speicher / Speicherbereich für die Interpolation 33
4.7 Graphische Darstellung der Rechenschritte in `COLINT` 34
4.8 Interpolation VHDL und Matlab verglichen 36

5.1 2D-DFT Architektute 38
5.2 Darstellung der komplexen Twiddle-Matrix in Real- und Imaginärteile . . 39
5.3 Verdrahtung der Komponente im Modul `1d_Mod` 40
5.4 Programmablauf der Komponente addUnit.vhdl 42
5.5 Zellbildung aus der Summe sequenzieller Produktteilen 43
5.6 Rechenablauf der Komponente OperatingUnit 45
5.7 Berechnungsschritte für die Bildung einer Speicherzelle 48

6.1 Ermittlung der Filterkoeffizienten über `Matlab-freqz2()` 49
6.2 Die sequentielle Paarbildung der Filterkoeffizienten in 32 Bit Speicherzelle 50
6.3 Programmablaufplan des 2D-Filters 52

7.1 Ergebnisse der VHDL-Interpolation im Vergleich mit der Matlab Implementierung 53
7.2 Ergebnisse der VHDL-Interpolation im Vergleich mit Octave- / Matlab-Implementierung 54
7.3 Ergebnisse der VHDL-Interpolation im Vergleich mit Octave- / Matlab-Implementierung 55
7.4 graphische Darstellung der Ergebnisse des Tiefpassfilers in VHDL 56
7.5 Erzeugte Standardzellen auf Xilinx FPGA-Ebene 57
7.6 Erzeugte Standardzellen auf Xilinx FPGA-Ebene 58
7.7 Erzeugte Standardzellen auf Cadence Chip-Ebene 59

1 Einleitung

Magnetische Sensoren sind heute in vielen Bereich der Technologie eingesetzt vor allem in der Automobilelektronik aufgrund ihrer berührungslosen und präzisen Erfassung von Drehzahl und Winkelinformation. Die HAW Hamburg in ihrer Zusammenarbeit mit Partnern an dem Forschungsprojekt „ISAR“ , das vom Bundesministerium für Bildung und Forschung gefördert wird.
Die vorliegende Masterarbeit leistet einen Teilbeitrag im Forschungsprojekt ISAR, das Signalverarbeitungsverfahren für magnetische Sensor-Arrays untersucht. In dem Projekt soll ein magnetisches Sensor-Array mit der dazu gehörigen Signalverarbeitung und der Systemarchitektur entwickelt werden.

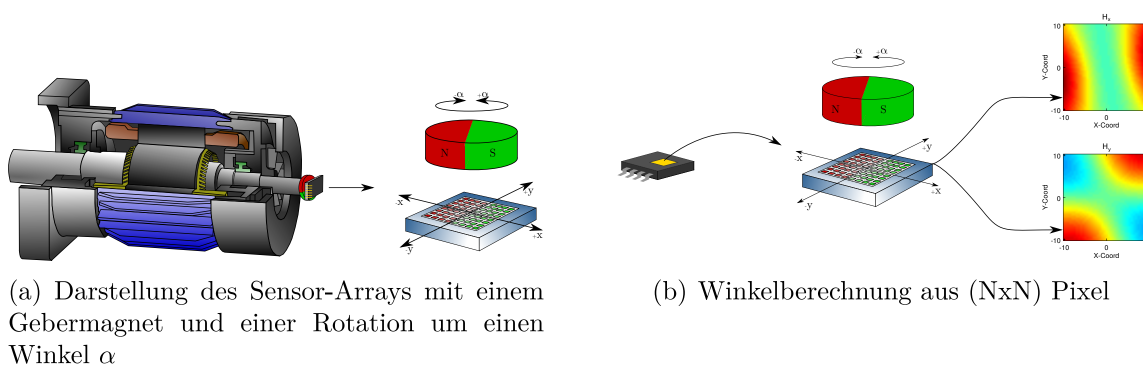
1.1 Stand der Technik

Heutzutage ist die Sensoranforderung in der technologischen Welt besonders in der Automobilindustrie stetig steigend. Sensoren sollen nicht nur kostengünstig sondern auch robust, präzise und energiesparend sein. Magnetische Sensoren werden in der Fahrzeugelektronik oft eingesetzt , da sie präzise Messdaten liefern. Diese Tendenz wird in Zukunft deutlich größer aufgrund des Übergangs der Automobilindustrie in die Integration von modernen `X-by-Wire`- und Regelungssystemen. Im Vergleich mit weiteren technologischen Produkten, sind magnetische Sensoren in ihren Einsatzbereichen vorteilhaft: eine akkurate und präzise Unterstützung bei den Automobilherstellern für die Winkelmessung, die Stabilität sowie die Sicherheit im Fahrzeug sind reibungsfrei gewährleistet. Weitere Anwendungsbereiche für magnetische Sensoren in modernen Technologien findet man in der Medizintechnik, Kameratechnik, Bewegungssteuerung, Robotik, Luftfahrt etc.
Die Signalverarbeitung in diese hochmodernen technologischen Gebieten dient der Verbesserung von Sensordaten und Gewinnung von Information. Die Signalverarbeitung wird sowohl in Hardware als auch in Software entwickelt. Abbildung 1.1 ist eine grobe Darstellung des Signalverarbeitungssystems [7].

Abbildung 1.1: Eine vereinfachte graphische Darstellung eines Signalverarbeitungssystems

1.2 Motivation

Magnetische Sensoren finden ihre Anwendung in der Automobilindustrie und besonders in der Motorelektronik und im Bremssystem aufgrund ihrer berührungslosen Erfassung von Drehzahlen und Winkelinformation. Die Hochschule für Angewandte Wissenschaften Hamburg (HAW) in ihrer Zusammenarbeit mit der Hochschule für Angewandte Wissenschaften Ostfalia forscht im Projekt `ISAR`: Signalverarbeitung für Integrated Sensor-Arrays basierend auf dem Tunnel-Magnetoresistiven Effekt für den Einsatz in der Automobilelektronik. Ziel dieses Projekts ist vom Entwurf bis zur Realisierung der Signalverarbeitung und der Systemarchitektur ein neuartiges Sensor-Array zu entwickeln. Mit einem Permanentmagnet ist es möglich mit dem Sensor-Array die Aufnahme der Winkelinformation des Gebermagnetfeldes durchzuführen. Durch Redundanzen und Ortsauflösung sollen Positionierung oder Systemfehler in einer digitalen Signalverarbeitung detektiert und kompensiert werden. Eine prinzipielle graphische Darstellung dieses Messprozesses illustrieren folgenden Abbildungen



(a) Darstellung des Sensor-Arrays mit einem Gebermagnet und einer Rotation um einen Winkel α

(b) Winkelberechnung aus (NxN) Pixel

Abbildung 1.2: Erfassung der Sensorinformation bei einem Permantenmagneten. Quelle: Projekt `ISAR` – HAW Hamburg

1.3 Ziel dieser Arbeit

In der vorliegenden Arbeit werden wichtige Module der Signalverarbeitung mit der Hardwarebeschreibungssprache `VHDL` (Very High Speed Integrated Circuit Hardware Description Language) implementiert. Diese Module umfassen die Interpolation, die zweidimensionale diskrete Fouriertransformation (2D-DFT) und eine 2D-Filterung. Die Interpolation wird linear implementiert. Sie wird aus quadratischen Eingangsdaten eines Sensor-Arrays der Größe (8×8) eine Datenmatrix desselben Typs und der Größe (15×15) als Resultat darstellen. Mit den interpolierten Sensordaten wird die Fehlerkorrektur verbessert und die Signalverarbeitung für ein neuartiges Sensor-Array kann realisiert werden. Anschließend wird mit einem Modul der zweidimensionalen-diskreten Fouriertransformation (2D-DFT), auf die Resultate der Interpolation zugegriffen und fortgesetzt. Um

irrelevante Informationen oder Störgrößen im Ortsraum zu beseitigen, wird eine 2D-Filterung als drittes Modul umgesetzt. Die Ortsfrequenzanteile, die nach Einsatz der 2D-Filterung übrig geblieben sind, werden bei der Berechnung der Nutzinformation verwendet. Anschließend wird das Gesamtsystem auf FPGA-Ebene mit den Sensordaten getestet. Damit wird gezeigt, dass bei den implementierten Funktionen keine Fehler bzw. numerische Probleme, wie zum Beispiel ein Überlauf, stattfinden.

2 Grundlagen

In diesem Kapitel wird auf die Grundlagen der digitalen Signalverarbeitung eingegangen, die für diese Arbeit benötigt werden. Zunächst wird das Zahlensystem in der Umgebung des Dualsystems betrachtet. Eine grobe Definition von ganzen Zahlen in der Binärdarstellung wird hier formal behandelt, gefolgt von der reellen Zahlen und ihre Arithmetik, deren wichtige Komponenten in der digitalen Signalverarbeitung die Gleit- und Festkommazahlen bleiben. Die hier zuletzt erwähnte Komponente der reellen binären Zahlen, die Festkomma-Darstellung, die entlang der Entwicklung des gesamten Systems angewandet wurde, wird in diesem Kapitel ausführlich erklärt. Anschließend auf die Veranschaulichung bestimmter Algorithmen eingegangen und ausführlich dargestellt, die in weiteren Kapiteln dieser Arbeit praktisch in VHDL entwickelt werden.

2.1 Zahlensysteme

Zur Realisierung digitaler Systeme verwendet man nur zwei Zustände.Daher werden nur zwei Ziffernsymbole betrachtet. Die Wahl der Basis 2 ist hier geeignet für die Zahlendarstellung. Eine Zahlendarstellung auf dieser Basis, das sogenannte *Dualsystem*, ist eine Ziffernfolge von Nullen und Einsen. Die wesentlichen Grundlagen für die Realisierung algorithmischer Aufgaben in digitalen Systemen sind die Zahlensysteme. In diesem Abschnitt werden die Formate der natürlichen und ganzen Zahlen sowie der Fest- und Gleitkommazahlen vorgestellt.

2.1.1 Natürliche Zahlen

In der Digitaltechnik sind natürliche Zahlen als Vorzeichenlose ganze Zahlen definiert. Der vorgesehene Zahlenbereich umfasst die positiven ganzen Zahlen inklusive der Null. Bei einer n Bit Zahlendarstellung ist der Zahlenbereich im Intervall $\left[0 - 2^{n-1}\right]$ eingeschränkt. Mathematisch lassen sich natürliche Zahlen allgemein wie folgt definieren:

$$Z = \sum_{i=-0}^{n-1} X_{\mathrm{i}} \cdot b^{i} \tag{2.1}$$

$$Z = X_{\mathrm{n-1}} \cdot b^{n-1} + ... + X_1 \cdot b^1 + X_0 \cdot b^0 \tag{2.2}$$

Dabei ist X_{i} ein Zahlenwert und b steht für die Basis. Für (b = 2) spricht man von Dualsystem, (b = 8) Oktalsystem, (b = 10) Dezimalsystem und für (b = 16) Hexadezimalsystem [4].

2.1.2 Ganze Zahlen (Zweierkomplement-Darstellung)

Unter ganzen Zahlen soll man die Erweiterung der natürlichen Zahlen verstehen um negative Zahlen darzustellen. Bei n Bit Zahlendarstellung, ergibt sich ein Zahlenbereich von -2^{n-1} bis $-2^{n-1}-1$. Die Codierung von negativen Zahlen wird anhand des sogenannten Vorzeichen-Betrag-Darstellung durchgeführt. Das Vorzeichen der negativen Zahl ist für das hochwertige Bit (MSB) vorgesehen. In der Regel ist die 0 für die Darstellung der positiven Zahlenwerte und die 1 soll das Vorzeichen der negativen Zahlenwerte darstellen. Die restlichen Bits stellen den Betrag der Zahl dar, welche als Vorzeichenlose Dualzahl codiert wird. Die Zweierkomplement-Darstellung ist eine Zahlendarstellung die folgende aufgelistete Punkte erfüllt:

- Der Zahlenwert Null ist einmal codiert
- Eine Additionsrichtung für den gesamten Zahlenbereich
- Sowohl Überlauf als auch Unterlauf treten an einer bestimmten Position auf

Alle Codierungen mit einer Führenden 1 sind im Zweikomplement als negative Zahlenwerte betrachtet. Mathematisch lässt sich die Zweierkomplement-Darstellung anhand der einfachen Summen-Formel angeben:

$$Z = -z_{N-1} \cdot 2^{N-1} + \sum_{i=0}^{N-2} z_i \cdot 2^i \tag{2.3}$$

Für eine Wortbreite von 4 Bit ist die Zahl -8 der am kleinsten codierte negative Zahlwert. Abbildung 2.2 zeigt für eine 4 Bit Wortbreite die möglichen darstellbaren Zahlen in Zweierkomplement in einem Zahlenkreis.

2.1.3 Darstellung reeller Zahlen

In der Mathematik werden Zahlen zu gewissen Zahlenmengen zugeordnet. Üblicherweise beginnt man mit den Natürlichen Zahlen, gefolgt von den ganzen, rationalen, reellen und den komplexen Zahlen. Die Unendlichkeit diese Zahlen macht ihre Darstellung im Rechner schwierig. Die bis jetzt gebauten Rechner verarbeiten nur endliche viele Stellen. Daher muss der darstellbare Zahlenbereich eingegrenzt werden. Somit werden Zahlen genau im Rechner dargestellt, die in dem Wertebereich zugelassen sind. Reelle Zahlen lassen sich auf zwei unterschiedlichen Weisen darstellen:

- Die Festkomma-Darstellung: Das Komma soll bei diesem Zahlenformat an einer fest vorgegebenen Stelle bleiben.
- Die Gleitkomma- oder Fließkomma-Darstellung: Das Komma so verschieben, dass nur signifikante Stellen erhalten bleiben [5].

Festkomma-Darstellung

Die Festkomma-Darstellung besteht aus folgenden Parametern und lässt sich durch diese einfache Gleichung beschreiben:

$$Z = (s, i, f) \tag{2.4}$$

Wobei:

- s: die Darstellung des Vorzeichens ist
- i: ein ganzzahliger Anteil repräsentiert
- f: der gebrochene Anteil ist

Der in Gleichung 2.5 stehende Bitvektor stellt die Zahl in Gleichung 2.6 in der Festkomma-Darstellung dar.

$$(X_{\mathtt{vk}-1}, ..., X_1, X_0, X_{-1}, ..., X_{-\mathtt{nk}+1}, X_{-\mathtt{nk}})_2 \tag{2.5}$$

$$Z = \sum_{i=-nk}^{vk-1} X_{\mathrm{i}} \cdot 2^i \tag{2.6}$$

vk und nk stehen für die Vorkomma- bzw. Nachkommastellen. In der binären Darstellung in Gleichung 2.5 liegt das Komma rechts der Stelle X_0.
Allgemein lässt sich eine Festkomma-Dualzahl der Wortbreite N mathematisch wie folgt darstellen:

$$Z = -z_{\mathrm{M}-1} \cdot 2^{M-1} + \sum_{i=-L}^{M-2} z_{\mathrm{i}} \cdot 2^i \tag{2.7}$$

Dabei ist M die Anzahl der Vorkommastellen und L die der Nachkommastellen. Die benötigte Wortbreite N einer solchen Zahlendarstellung ist:

$$N = M + L \tag{2.8}$$

Addition im Dualsystem

Die Addition zweier Festkommazahlen A und B im Dualsystem lässt sich genauso wie im Dezimalsystem durchführen.

$$Z = A + B \tag{2.9}$$

- Z: Summe
- A: Summand
- B: Summand

Die Addtion erfolgt ausgehen von der niederwertigen zur höherwertigen Stelle, unter Berücksichtigung des Übertrags. Addiert man die zwei Zahlen A und B der Wortbreite `n` so ist das Ergebnis Z maximal (`n + 1`) Bit lang.

Tabelle 2.1: Addition im Dualsystem

a_n	b_n	c_{n-1}	c_n	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Die Stellen a_n, b_n und der Übertrag c_{n-1} der vorher durchgeführten Stellenaddition werden zusammengefasst. Tabelle 2.1 zeigt die Addition zweier Dualzahlen der Wortbreite 2 und den Übertrag.

Bei der Addition zweier Festkommazahlen ist die Vorgehensweise identisch zum Dezimalsystem. Die Kommata stehen übereinander und es wird stellenweise addiert. In dem folgenden Beispiel ist das Prinzip der Addition bei Fließkommazahlen deutlich illustriert [2].

	0 1 1 0 0,0 1 0
	+ 0 0 1 1 0,1 1 1
Übertrag	1 1 0 0 1 1 0
	= 1 0 0 1 1,0 0 1

Abbildung 2.1: Addition zweier 8 Bit langen Festkommazahlen

Einerkomplement-Darstellung

Um das Einerkomplement einer Zahl im Dualsystem zu bilden, müssen bitweise alle Nullen gegen Einsen vertauscht werden und umgekehrt. In mancher Literatur ist die symbolische Darstellung des Einerkomplements einer Dualzahl A als $\bar{A}$ angenommen. Wenn A eine Dualzahl der Wortbreite `n` Bit ist, gilt:

$$\bar{A} + A = 2^n - 1 \tag{2.10}$$

Anhand der Gleichung 2.10 lässt sich das Einerkomplement einer Zahl durch eine einfache Umformung der Gleichung bestimmen:

$$\bar{A} = 2^n - 1 - A \tag{2.11}$$

Zweierkomplement-Darstellung

Die Bildung des Zweierkomplements A_{K2} einer Dualzahl A lässt sich durch eine anschließende zusätzliche Addition des Einerkomplements $\bar{A}$ mit 1 durchführen: Gleichung

2.12.

$$A_{\mathrm{K2}} = \bar{A} + 1 \tag{2.12}$$

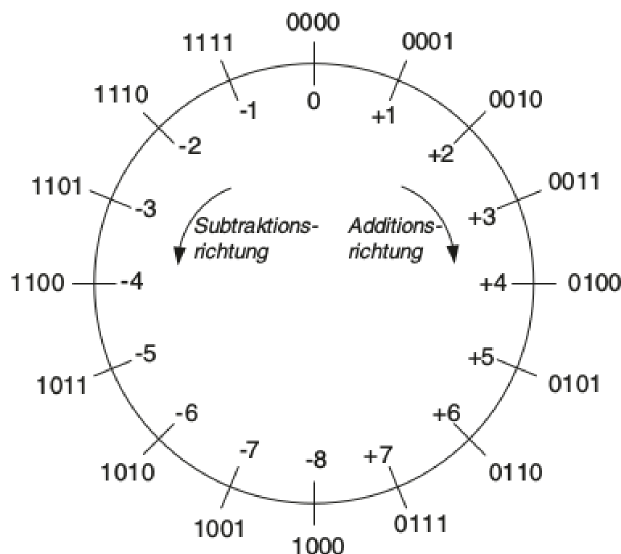


Abbildung 2.2: Zahlenkreis für 4 Bit Zahlen in Zweierkomplement-Darstellung [3]

Für eine n Bit Wortbreite ist der Zahlenbereich des Zweikomplements $[-2^{n-1}, 2^{n-1}-1]$. Die kreisförmige Darstellung für eine 4 Bit breite Dualzahl in Abbildung 2.2 verdeutlicht den Zahlenbereich. Aus der Darstellung lässt sich einfach entnehmen, dass es mehr negativen Zahlen gibt als positive. Die Anordnung des Zahlenbereich ist somit unsymmetrisch.
Durch Umformung und Ersetzung des Wertes von $\bar{A}$ aus Gleichung 2.11, erhält man anhand der folgenden Gleichung A_{K2} in Abhängigkeit von A:

$$A_{\mathrm{K2}} = 2^n - A \tag{2.13}$$

Subtraktion in Zweierkomplement-Darstellung

Werden zwei Dualzahlen A and B voneinander subtrahiert, kann man die Operation durch Verwendung des Zweierkomplements in Gleichung 2.13 durchführen.

$$A - B = A - B + B_{\mathrm{K2}} - B_{\mathrm{K2}} \tag{2.14}$$

Ersetzt man B_{K2} durch seinen Wert in Abhängigkeit von B

$$B_{\mathrm{K2}} = 2^n - B \tag{2.15}$$

dann wird aus Gleichung 2.14

$$A - B = A - B + B_{\mathrm{K2}} - (2^n - B) \tag{2.16}$$

$$A - B = A + B_{\mathrm{K2}} - 2^n \tag{2.17}$$

Man muss zum einen beachten, dass die binäre Schreibweise von $2^{\mathtt{n}}$, $(\mathtt{n}+1)$ Stellen benötigt und zum anderen vorsichtig sein im Zusammenhang mit einer Bereichsüberschreitung oder Überlauf (Overflow).
Die Bereichsüberschreitung bei der Addition von zwei Dualzahlen findet nur statt, wenn die beteiligten Operanden dasselbe Vorzeichen haben, entweder beide (+) oder beide (-).
In allen weiteren Fällen ist die Bereichsüberschreitung ausgeschlossen.

Tabelle 2.2: Additionsüberlauf der n Bit Zweierkomplement Darstellung

Arithmetik	Richtiges Ergebnis	Überlauf
$A + B$	$c_{\mathrm{n}} = 0,\ c_{\mathrm{n-1}} = 0$	$c_{\mathrm{n}} = 0,\ c_{\mathrm{n-1}} = 1$
$A - B$	$c_{\mathrm{n}} = c_{\mathrm{n-1}}$	nicht möglich
$-A - B$	$c_{\mathrm{n}} = 1,\ c_{\mathrm{n-1}} = 1$	$c_{\mathrm{n}} = 1,\ c_{\mathrm{n-1}} = 0$

Die Addition zweier Dualzahlen der Wortbreite `n` wird als richtig bewertet, wenn die Übertragstellen c_{n} und $c_{\mathrm{n-1}}$ des Ergebnis übereinstimmen. Tabelle 2.2 zeigt eine zusammengefasste Bewertung zweier arithmetischer Operationen im Dualsystem.

Multiplikation

Die Multiplikation zweier `n`-Bit positiven Dualzahlen liefert als größtes zu erwartende Ergebnis `E`:

$$\mathtt{E} = (2^{\mathtt{n}} - 1) \cdot (2^{\mathtt{n}} - 1) = 2^{2\mathtt{n}} - 2^{\mathtt{n}+1} + 1 \leq \ 2^{2\mathtt{n}} - 1 \tag{2.18}$$

Zwei `n`-Bit Dualzahlen multipliziert ergibt ein Ergebnis der Wortbreite `2n`-Bit.
Für die Multiplikation oder Division in Zweierkomplement-Darstellung von Dualzahlen bietet sich unter anderem folgender Algorithmus. Bei dem Verfahren geht es darum im ersten Schritt die vorzeichenlosen Zahlen oder Beträge der beteiligten Operanden zu berechnen, anschließend wird die bitweise Operation durchgeführt und als letzter Schritt wird das Ergebnis negiert, wenn die Operanden unterschiedliche Vorzeichen aufweisen.
Bei Festkommazahlen wird die Multiplikation, wie in der dezimalen Darstellung, ohne Berücksichtigung des Kommas durchgeführt.
Die Multiplikation zweier Festkomma-Dualzahlen `A` und `B` mit `m` bzw. `l` Nachkommastellen liefert als Ergebnis eine Festkomma-Dualzahl `C` mit `m+l` Nachkommastellen.

Gleitkomma-Darstellung

Eine detaillierte Beschreibung dieser Zahlendarstellung wird im Rahmen dieser Masterarbeit begrenzt, da die einzelnen Rechenvorgehensweisen anhand eines anderen Zahlenformats, nämlich der Festkomma-Zahl durchgeführt wurde. Daher wird im Folgenden

hauptsächlich das Grundprinzip der Gleitkomma-Darstellung eingegangen. Die Beschreibung eines Zahlenwertes in einer Gleitkomma-Darstellung lässt sich durch eine Mantisse M und einen Exponenten E darstellen. M und E werden hierbei als ganze Zahlen codiert und repräsentieren der Reihe nach den Vorzeichen-Betrag und die Bias-Darstellung. Ein Vorzeichenbit S ist zusätzlich vorgesehen. Die mathematische Gleichung eines Zahlenwertes Z_{GK} als Gleitkommazahl lässt sich wie folgt darstellen:

$$Z_{\text{GK}} = (-1)^S \cdot M \cdot 2^E \tag{2.19}$$

2.2 Prinzip der Interpolation nach Matlab-interp2()

Die Grundidee der Interpolation bei der Masterthesis ist aus dem Matlab-interp2()-Algorithmus entnommen. Diese Funktion des Matlab-Entwicklungstools erweitert eine quadratische Matrix A der Größe $(N \times N)$ zu einer weiteren quadratischen Matrix B der Größe $((2N-1) \times (2N-1))$. In der Berechnung sind dabei zwei Schritte umgesetzt:

- Zwischen zwei Werte in jeder Zeile der Matrix A, die in die Matrix B übernommen werden, wird noch ein weiterer Wert eingefügt, welcher dem arithmetischen Mittelwert der beiden übernommenen Werte entspricht. Damit eine Zeile aus der Matrix A mit N Elementen in der Matrix B Zeile 2N-1 Elemente lang. Das Parameter `N` ist hier als positive ganze Zahl zu betrachten. Das Prinzip der Berechnungschritte erfolgt linear

- Zwischen jede Zeile der Matrix A, die wie zuvor erläutert in die Matrix B übernommen wird, wird noch eine weiterere Zeile eingefügt. Diese eingefügte Zeile wird elementweise mit dem arithmetischen Mittelwert, den Elementen darüber und darunter, gefüllt.

Damit erfolgt eine lineare Interpolation in zwei Dimensionen. An den Rändern der Matrix wird nicht interpoliert, Randwerte tragen nur den interpolierten Zwischenwert welche im Innern der Matrix angeordnet sind. Elemente in der Mitte tragen zu allen umgebenden Zwischenwerten bei.

Das bewirkt, dass die Matrixgröße nicht in jeder Dimension verdoppelt wird `(2N)`, sondern um einen Wert in der Größe darunter ansteigt `(2N-1)`.

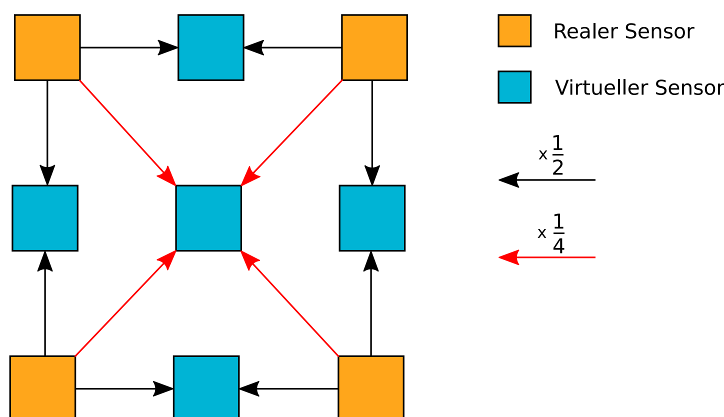


Abbildung 2.3: Ermittlung von interpolierten Sensordaten mittels Matlab_interp2(). Quelle: BA-Rindelaub HAW-Hamburg

2.3 Diskrete Fouriertransformation (DFT)

In der Welt der digitalen Signalverarbeitung und Signaltheorie ist die Diskrete Fouriertransformation eines der wichtigen Werkzeuge zur Analyse und Synthese von Signalen und Systemen mittels der Transformation diskreter Signalfolgen von Bildbereich zu Spektralbereich. Die Wichtigkeit des Algorithmus liegt darin, dass die DFT über eine schnelle Berechnungsvorgehensweise verfügt, die schnelle Fourier Transformation, (FFT) zur Berechnung von Spektralfolgen. Praktische Anwendungsbereiche der DFT finden unter anderem in Folgendem statt:

- In der Messtechnik
- In der digitalen Bildverarbeitung und Mustererkennung
- In der digitalen Signalübertragung mittels Mehrträgerverfahren wie z.B. OFDM (Orthogonal Frequency Division Multiplexing)
- In der digitalen Mobilfunksystemen
- In der Spektralanalyse sowie in der Medizintechnik zur Analyse von EEG-Signalen (EEG-Elektronenzephalografie)

In der digitalen Signalverarbeitung werden unter anderem diskrete Signalfolgen betrachtet, die vorher aus abgetasteten kontinuierlichen Signalen kommen. Bei der vorliegenden Arbeit werden finite Signalfolgen derart:

$$x(0), x(1), ..., x(\mathtt{N}-1) = x(\mathtt{k})_{0 \leq \mathtt{k} \leq \mathtt{N}-1} \tag{2.20}$$

bearbeitet. Das Signal ist N Signalwerten lang und wird anhand der Diskreten Fouriertransformation in eine Spektralfolge der Form:

$$X(0), X(1), ..., X(\mathtt{N}-1) = X(l)_{0 \leq \mathtt{l} \leq \mathtt{N}-1} \tag{2.21}$$

umgewandelt. Wie in Bildbereich, ist das Signal im Spektralbereich N Signalwerte lang. `k` und l in Gleichungen 2.20 und 2.21 sind ganzzahlige Laufindexe in Bild- bzw. Spektralbereich. Abbildung 2.4 stellt eine grobe graphische Darstellung des Transformationssystems dar [1].

Abbildung 2.4: Diskrete Transformation der Signalfolge x(k) in die Spektralfolge X(l)

2.3.1 Definition der Zweidimensionalen Diskreten Fouriertransformation (2D-DFT)

Die `DFT` ist nicht nur für eindimensionale Signalfolgen definiert, sondern für Funktionen unterschiedlicher Dimensionen. Somit sind zweidimensionale Funktionen wie die in der anliegenden Arbeit von besonderer Bedeutung.
Allgemein ist die `2D-DFT` eine Zweidimensionale periodische Funktion `f(u, v)` der Größe $(M \times N)$ wie folgt definiert:

$$F(m,n) = \frac{1}{\sqrt{M \cdot N}} \cdot \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} f(u,v) \cdot e^{-i \cdot 2\pi \cdot \frac{mu}{M}} \cdot e^{-i \cdot 2\pi \cdot \frac{nv}{N}} \tag{2.22}$$

Aus dem Produkt der Exponentialfunktionen ergibt sich die endgültige mathematische Formeldarstellung wie folgt

$$F(m,n) = \frac{1}{\sqrt{M \cdot N}} \cdot \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} f(u,v) \cdot e^{-i \cdot 2\pi \cdot (\frac{mu}{M} + \frac{nv}{N})} \tag{2.23}$$

Das Resultat im Spektralbereich `F(m, n)` ist ebenso wie die Anfangsfunktion `f(u, v)` in Bildbereich $(M \times N)$ groß.

2.3.2 Die Matrixrepräsentation der 2D-DFT

Für die Darstellung der diskreten Fourier Transformation wird ein Drehfaktor definiert.

$$W_N = e^{\frac{-j \cdot 2 \cdot \pi}{N}} = cos(\frac{2 \cdot \pi}{N}) - j \cdot sin(\frac{2 \cdot \pi}{N}) \tag{2.24}$$

unter Berücksichtigung der folgenden Gleichungen

$$e^{\frac{-j \cdot 2 \cdot \pi \cdot k \cdot l}{N}} = W_N^{k \cdot l} \tag{2.25}$$

$$e^{\frac{j \cdot 2 \cdot \pi \cdot k \cdot l}{N}} = W_N^{-k \cdot l} \tag{2.26}$$

lässt sich die DFT-Transformationsgleichungen wie folgt darstellen

$$X(l) = \sum_{k=0}^{N-1} x(k) \cdot W_N^{k \cdot l} \tag{2.27}$$

$$x(k) = \frac{1}{N} \sum_{k=0}^{N-1} X(l) \cdot W_N^{-k \cdot l} \tag{2.28}$$

mit der Umformung der Gleichung 2.27

$$X(l) = \sum_{k=0}^{N-1} x(k) \cdot W_N^{k \cdot l} = \sum_{k=0}^{N-1} W_N^{k \cdot l} \cdot x(k) \tag{2.29}$$

$$= W_N^{l \cdot 0} \cdot x(0) + W_N^{l \cdot 1} \cdot x(1) + ... + W_N^{l \cdot (N-1)} \cdot x(N-1) \tag{2.30}$$

$$= (W_N^{l \cdot 0}, W_N^{l \cdot 1}, ..., W_N^{l \cdot (N-1)}) \cdot \begin{bmatrix} x(0) \\ x(1) \\ . \\ . \\ . \\ x(N-1) \end{bmatrix} \tag{2.31}$$

mit

$$x = \begin{bmatrix} x(0) \\ x(1) \\ . \\ . \\ . \\ x(N-1) \end{bmatrix} \tag{2.32}$$

$$X(l) = (W_N^{l \cdot 0}, W_N^{l \cdot 1}, ..., W_N^{l \cdot (N-1)}) \cdot x \tag{2.33}$$

mit dem Laufindex l aus dem Intervall [0, N-1], lässt sich die finite Spektralfolge X(l) wie folgt ausdrucken

$$X = W_{\mathrm{N}} \cdot x \tag{2.34}$$

mit

$$W_N = (w_N^{lk})_{0 \leqslant k, l \leqslant N-1} \tag{2.35}$$

2.4 Das 2D-Digitalfilter

Das Digitalfilter ist ein fundamentaler Bestandteil zur Gewinnung relevanter Informationen in der digitalen Signalverarbeitung. Das Analoge Signal wird mittels eines Analog-Digital-Konverters (ADC) digitalisiert und kann anhand bereits entwickelten Tools, von Computer weiterverarbeitet werden. Eine allgemeiner Signalflussplan des analogen Eingangssignals und die darauf folgende Umwandlung in die digitale Welt wird in der folgenden Abbildung schematisiert [8].

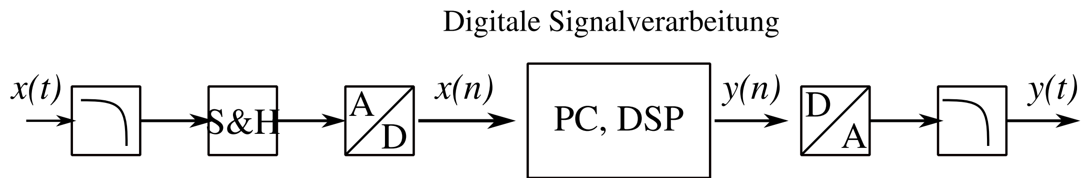


Abbildung 2.5: Ablaufplan der Digitalisierung analoger Signale mittels PC oder Digital Signal Processing (DSP)

2.4.1 Grundstruktur linearer digitaler Filter

Angenommen $x(n)$ sei ein digitales Signal. Als digitales Filter wird jedes System bezeichnet, das aus dem Signal $x(n)$ nach bestimmten mathematische Gesetzen ein Signal $y(n)$ erzeugt, ohne dabei den Zeitbereich verlassen zu müssen.

Abbildung 2.6: Signalverlauf mit digitalem Filter

Die Ein- und Ausgangssignale $x(n)$ bzw. $y(n)$ haben die identischen physikalischen Größen als unabhängige Variable. Aus den beiden Signalen lässt sich die allgemeine Form einer linearen Differenzengleichung der Ordnung `k` darstellen.

$$y(n) = \sum_{i=0}^{k} d_i \cdot x(n-i) - \sum_{i=1}^{k} g_i \cdot y(n-i) \tag{2.36}$$

- k ist die Anzahl der Verzögerungsglieder für das Eingangssignal und Ausgangssignal
- g_i und d_i sind Filterkoeffizienten

Das Ausgangssignal eines solchen Filters ergibt sich als Linearkombination der Filterkoeffizienten und verzögerten Ein-und Ausgangsgrößen. Vorteil dabei sind die geringe Elementaroperationen wie die Addition und Multiplikation, die durchgeführt werden. Ein digitales wird `nichtrekursiv` oder auch `FIR-Filter` bezeichnet, wenn der linke Teil der Gleichung 2.36 vorhanden ist. Die Dauer der Impulsantwort dieses Filters ist kürzer als k+1 Abtastperioden.
Ist aber nur der rechte Teil der Gleichung vorhanden, spricht man von einem `rekursiven` (Rückkopplung) oder `IIR-Filter`

2.4.2 Tiefpassfilterung im Frequenzbereich

Behandelt wird in diesem Abschnitt nur die Filterung im Frequenzbereich, ein zweidimensionales Digitalfilter `(2D-Filter)`. Das digitale Filter, das in dieser Arbeit entworfen

und implementiert wird, soll aus einem 2D-Eingangssignal, also $(N \times N)$ Matrixeingang, ein Ausgangssignal derselben Dimension unter Verwendung von zuvor berechneten Koeffizienten des Gauß-Tiefpassfilters bestehen.
Im Gegensatz zum rechteckigen Filter besitzt die Gaußsche Glocke eine spezielle Gestalt für die Behandlung der Frequenzen. Ein weiterer Vorteil dieses Filters ist, dass die Fourier-Transformierte eine Gauß-Form hat. Allgemein ist die mathematische Übertragungsfunktion des Gauß-Tiefpassfilters im Frequenzbereich wie folgt ausgedrückt.

$$H(u,v) = e^{-\frac{-D^2(u,v)}{2\sigma^2}} \tag{2.37}$$

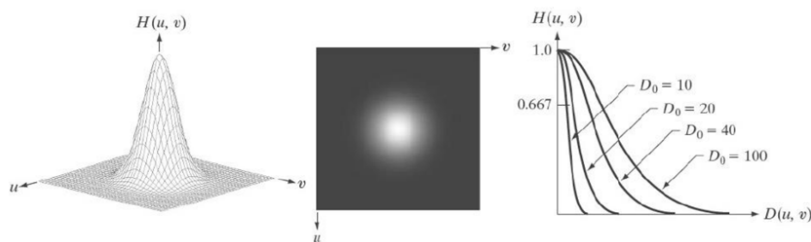


Abbildung 2.7: Gauss-Tiefpassfilter nach [6]

Die Parameter `u, v` in der Gleichung der Übertragungsfunktion des Filters `H(u,v)`, stehen als Koordinaten des Bildes im Frequenzbereich. σ steht für die Standardabweichung und ist einheitslos.
Ein Ausgangssignal des Gauß-Tiefpassfilters lässt sich dann laut Abbildung 2.8 aus Gleichung 2.38 mathematisch errechnen.

Abbildung 2.8: Gauss-Tiefpassfilter Bildverarbeitung Prof. Kölzer HAW-Hamburg

$$B = A.F \tag{2.38}$$

- A Eingangsarray $(N \times N)$
- A Filterkoeffizienten als Matrix der Größe $(N \times N)$
- B Ausgangsarray $(N \times N)$

Das Matrixprodukt wird zellenweise ausgeführt, $b_{ij} = a_{ij} * f_{ij}$.

3 Entwicklungskonzept

In dem Kapitel geht es hauptsächlich darum die Entwicklungsvorgehensweise detailliert wie möglich aufzuteilen. Das Entwicklungskonzept wird hier schrittweise beschrieben. Vom gesamten Aufbau des Systems bis zum Entwurf der einzelnen Bearbeitungsschritten werden kurz und kompakt behandelt. Die in Abbildung 3.1 dargestellter Aufbau, wird entlang der gesamten Entwicklung für die Implementierung der in der Arbeit entwickelten Modulen genutzt. Es besteht aus dem ZedBoard der Firma Xilinx und dem Tiva-Board von Texas Instruments. Eine detaillierte Beschreibung dieser Aufbau findet man in der Bachelorarbeit von Herr Helck BA-HAW-Hamburg.

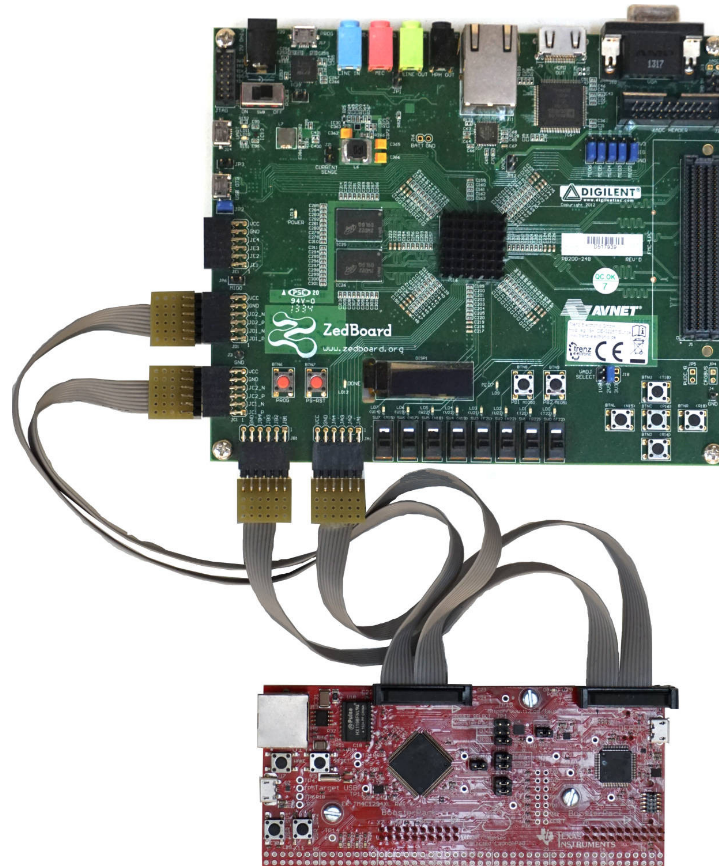


Abbildung 3.1: Die Testplattform besteht aus dem Mikrocontroller TM4C1294 auf dem Evalutionsboard Connected Lauchpad von Texas Instruments und dem FPGA-Zedboard von Xilinx

3.1 Systementwurf auf dem FPGA

Für die Systementwicklung wird ein Block RAM (BRAM) eingesetzt. Dieser wird für die Speicherung der Zwischenergebnisse genutzt. Der BRAM ist $(2^{10} \times 32Bit)$ groß und bekommt als erstes, bevor ein Modul eingeschaltet wird, die komplexen Rohdatenmatrix (Sensordaten) $(8 \times 8 \times 32Bit)$, die $(7 \times 7 \times 32Bit)$ Twiddle-Faktoren und die $(120 \times 32Bit)$ Filterkoeffizienten über den Mikrocontroller geschrieben (Abbildung 3.1). An BRAM sind verschiedene Module angeschlossen. In dieser Arbeit werden die Module aus Abbildung 3.2 beschrieben. Dazu gehören die Interpolation, die 2D-DFT sowie die 2D-Filterung.

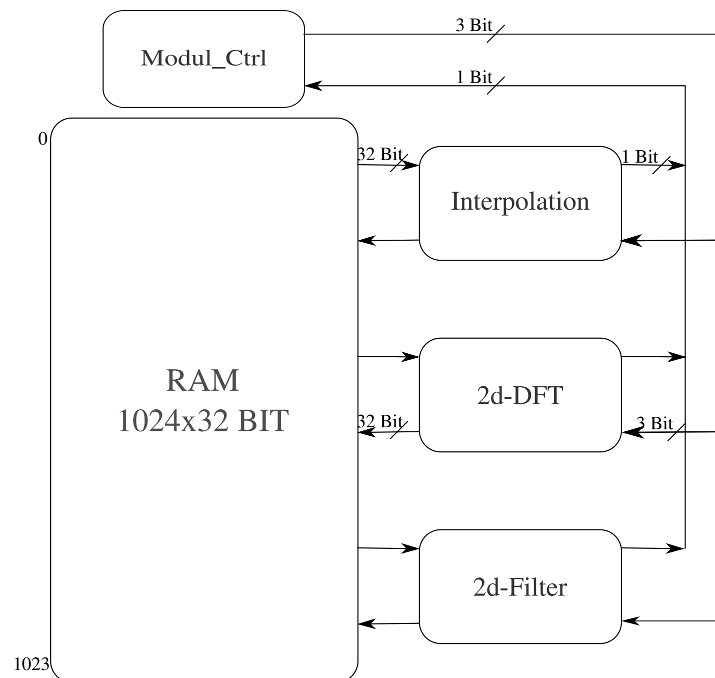


Abbildung 3.2: Systementwurf auf FPGA

Um die Verarbeitung der Sensordaten effizient zu gestalten, wurde ein Modul entworfen, das die Steuerung im gesamten System übernimmt, hier die Rolle von `Modul_Ctrl`. Seine Hauptaufgabe in dem gesamten System ist das Ein- und Ausschalten von weiteren Modulen, die im System vorhanden und an den BRAM angeschlossen sind. Das `Modul_Ctrl` hat die Rolle eines Masters analog zu einem Master-Slave-System. Über sein `3 Bit` Ausgangssignal vom Typ `std_logic_vector(2 downto 0)`(mit maximal 8 Modulen) werden die nacheinander liegenden Module sequentiell angesprochen. Zur Aktivierung des Interpolationsmoduls ist der Wert des Signals `"100"` aufweisen. Damit werden automatisch alle weitere Module, die an den BRAM und das `Modul_Ctrl` angeschlossen sind `hochohmig` geschaltet. Sie können an am BRAM weder Lese- noch Schreibaktionen durchführen. Sobald die Berechnungen im aktivierten Modul beendet sind, wird am Ausgang des Moduls ein `ready-Signal` aktiv, das als Eingang im `Modul_Ctrl`

vorgesehen ist.
Mit dem aktiven `ready-Signal`(1 Bit `std_logic`) zeigt dem `Modul_Ctrl` an, dass das vorher aktive Modul seine Aufgabe beendet hat.Somit kann das nächste stehende Modul eingeschaltet werden. Nach diesem Prinzip werden die unterschiedlichen Aufgaben der Module durchgeführt. Ab Kapitel 4 werden die einzelnen Aufgaben der in Abbildung 3.2 dargestellten Module ausführlich erläutert.

3.2 Entity des Moduls

Die Entity ist die externe Beschreibung des Moduls. Unter der `Entity` eines Moduls sind die Schnittstellen des Moduls zu verstehen, die einzelnen Ein- und Ausgangssignale worüber das Modul verfügt, um mit der Außenwelt zu kommunizieren. Aus der groben Systemdarstellung in Abbildung 3.2, lässt sich entnehmen, dass die Kommunikationsleitungen bei allen drei Signalverarbeitungsmodulen identisch sind. Sowohl das Modul `Interpolation` als auch die zwei weiteren Module `2D-DFT` und das `2D-Filter`, nutzen die gleichen Leitungen, um zum einen Daten aus dem Speicher zu lesen, bzw. Daten in den Speicher zu schreiben und zum anderen auf Befehle aus dem `Modul_Ctrl` zu reagieren. Eine nähere Betrachtung eines einzelnen Moduls ist in Abbildung 3.3 dargelegt.

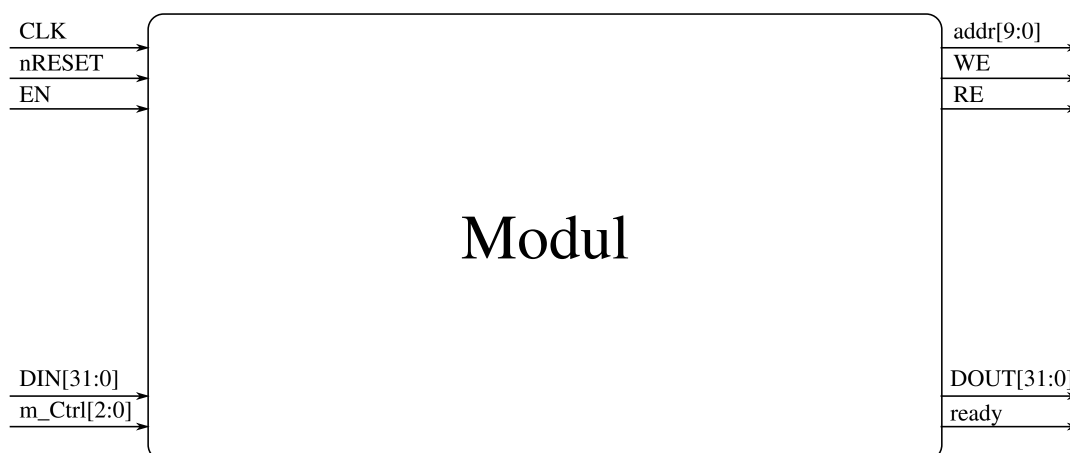


Abbildung 3.3: Allgemeine Schnittstellen der am `BRAM` angeschlossenen Signalverarbeitungsmodule

Die Schnittstellen sind in zwei Gruppen unterteilt:

- Die Eingänge (Inputs):
  - Die Steuersignale:
    - `CLK`: Taktsignal für die Synchronisation des Systems
    - `nRESET`: löschen aller Ausgänge gelöscht und zurücksetzen interner Signale

      * EN (ENABLE): Aktivierung des Moduls
    - DIN[31:0], 32 Bit breiter Dateneingang mit Verbindung zum BRAM
    - m_Ctrl[2:0] 3 Bit breit und enthält Befehle (Werte) zur Aktivierung des Moduls. Da Signal verbindet Modul_Ctrl mit den angeschlossenen Modulen

- Die Ausgänge (Outputs):
    - addr[9:0]: ist 10 Bit breit. Damit werden die einzelnen Speicherzellen des BRAMs angesprochen
    - WE: Write_Leitung zur Aktivierung des Schreibvorgangs
    - RE: Read_Leitung zur Aktivierung des Lesevorgangs. WE als auch RE verbinden Speicherbaustein mit dem Modul
    - DOUT[31:0]: Das 32 Bit Ausgangssignal ermöglicht das Schreiben der Ergebnisse in den BRAM. Das Signal wird hauptsächlich während des Schreibvorgangs eingesetzt
    - ready: Sobald das Modul mit seiner Aufgabe fertig ist, wird das Signal ready aktiviert. Es ist neben dem Signal m_Ctrl[2:0], das als Input vorgesehen ist, die zweite Leitung, die mit dem Modul_Ctrl verbunden ist

Tabelle 3.1: Beschreibung der Signale für die Eingänge und Ausgänge.

Eingänge (Inputs)		Ausgänge (Outputs)	
CLK	Taktsignal für die Synchronisation des Systems	addr[9:0]	10 Bit breiter Vektor zum Ansteuern der Speicherstellen des RAM
nRESET	Signal zum zurücksetzen aller Signale	WE	Signal zur Aktivierung der Schreibe-Aktion.
EN	Signal zur Aktivierung des ausgewählten Moduls.	RE	Signal zur Aktivierung der Lese-Aktion.
DIN[31:0]	Eingangsdaten (16 Bit Real, 16 Bit Imaginär)	DOUT[31:0]	Ausgangsdaten (16 Bit Real, 16 Bit Imaginär)
m_Ctrl[2:0]	Auswahlvektor für die Module.	ready	Signal für das Bearbeitungsende eines Moduls.

3.3 Der System-Multiplexer

Der System-Multiplexer ist kein eigenes Modul, sondern eine Implementierungsart im System, die es ermöglicht, das über das Modul_Cltr selektierte Modul mit Priorität einzusetzen. Anhand des System-Multiplexers werden Signalkonflikte vermieden, indem alle

weiteren Modulschnittstellen des Systems hochohmig gesetzt werden, außer das aktuelle ausgewählte Modul ist berechtigt, Daten aus dem Speicher zu holen, die Berechnungen zu starten und die Ergebnisse zu schreiben. Die selbe Implementierungsart kann innerhalb eines Moduls umgesetzt werden, wenn dies als Toplevel weitere Komponente ist. Abbildung 3.4 zeigt ein Beispiel für den Multiplexer, wie er in dieser Arbeit eingesetzt wird. Die Darstellung zeigt den Aufbau zwei verschieden Komponente innerhalb eines Moduls also die Architektur eines Moduls als Toplevel zweier Komponenten. Bei dem Punkt ist es wichtig zu erwähnen dass, für denselben Zweck, außerhalb den Modulen, werden die Read- und Writeleitung des Systems über `NOR-Gatter` verknüpft und weiter zum BRAM verdrahtet. Diese Entwicklungsprozess wird in der Bachelorarbeit von Herr Helck ausführlich detailliert.

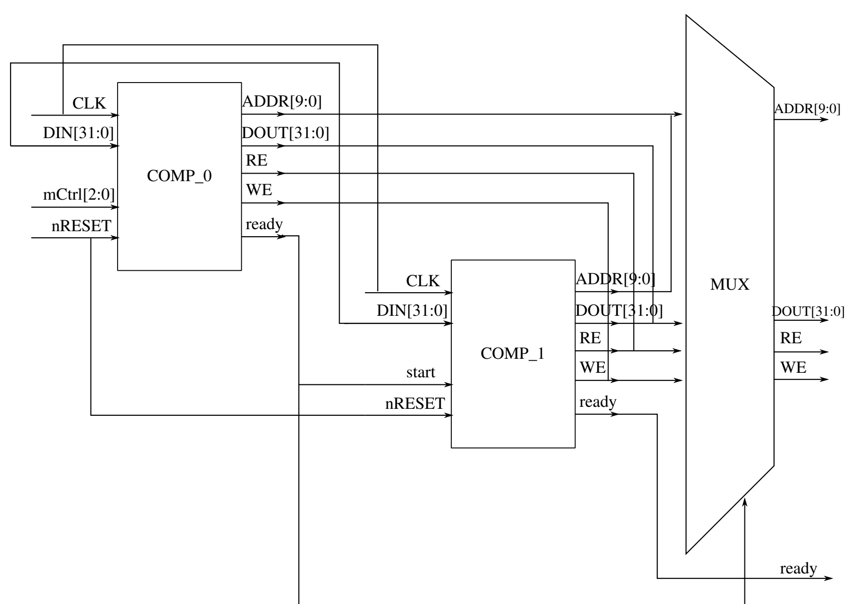


Abbildung 3.4: System-Multiplexer in Modulbeschreibung

3.4 Speicherverwaltung

Damit die Daten akkurat im Speicher hinterlegt werden, muss zuvor geklärt werden, welche Datenmenge jedes Modul nach der Datenverarbeitung erzeugt. Die Datenmenge der komplexen Eingangsmatrix ist gegeben. Sie ist $(8 \times 8 \times 32Bit)$ groß und befinden sich in den ersten oberen 64 Zellen des `BRAM`. Diese Daten werden im Laufe der Verarbeitungszeit in den weiteren Modulen Algorithmen durchlaufen und ändern dabei sowohl den Beträge als auch die Große des Datenpakets.

Abbildung 3.5 zeigt eine graphische Repräsentation des Signalflussplans der Datenverarbeitung in der vorliegenden Arbeit.

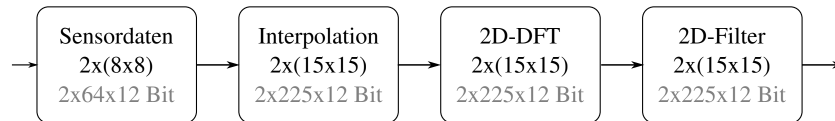


Abbildung 3.5: Signalflussplan der Verarbeitung der Sensordaten

Der Faktor `2` vor der Matrizenschreibweise steht für die komplexe Darstellung der Daten. Bei jedem Algorithmus werden $(n \times n)$ reale-und imaginäre Daten generiert. Abbildung 3.6 zeigt die Anordnung der Daten im BRAM.
Aus der Abbildung lassen sich folgende Merkmale entnehmen. Es werden die Ergebnisse von den 2D-Diskreten Fouriertransformationen (`DFT`) im BRAM abgespeichert. Laut Gleichung 5.1 wird zunächst die `1D-DFT` berechnet und mit den Ergebnissen die Daten der `2D-DFT` ermitteln. Ein externer Zwischenspeicher ist dafür wichtig, denn die Datenmatrix würde für eine interne Speicherung der `1D-DFT` $(225 \times 32Bit)$ einen großen Flächenbedarf auf Chipebene beansprucht. Da bei der Chipentwicklung vermieden werden soll, Datenmengen dieser Größe(für die `1D-DFT`: $15 \times 15 \times 32Bit$) lokal abzuspeichern aufgrund ihres Flächenbedarfs, sollte dann bei der Speicherverwaltung eine Speichergröße maximal von $(225 \times 32Bit)$ für die `1D-DFT` vorgesehen werden. Sie ist als Zwischenergebnis des 2D-DFT_Moduls zu betrachten.
Der restliche Speicherplatz ist für das `2D-Filter` Modul nicht ausreichend. Daher müssen die Ergebnisse dieses Moduls entweder im Speicherbereich der Interpolation oder der 1D-DFT überschrieben werden.

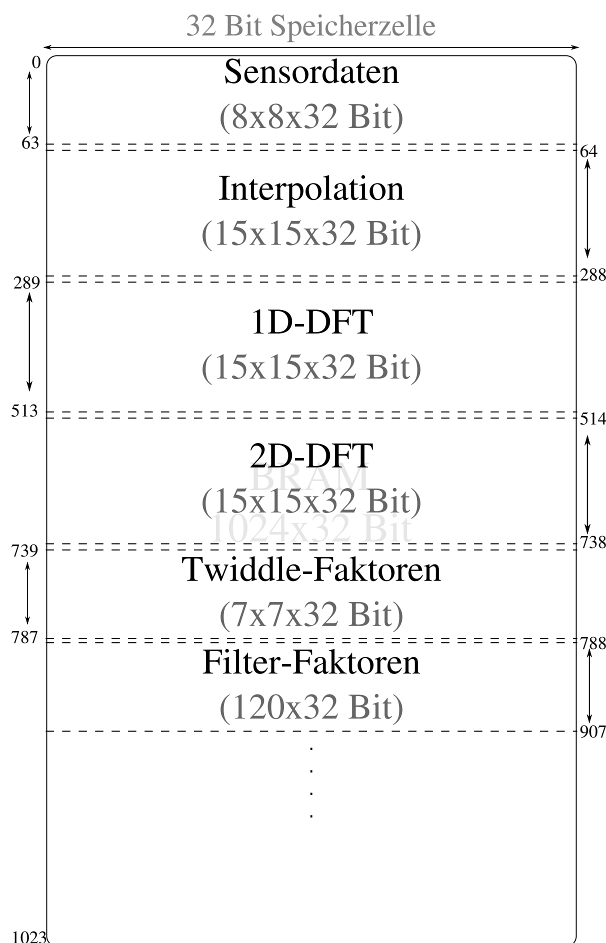


Abbildung 3.6: Speicherverwaltung und Datenaufteilung im `BRAM Speicher`

3.5 Datenformat im Speicher

Die vorliegende Arbeit behandelt die Speicherung komplexe Daten. Die Sensordaten, die über den Mikrocontroller in einem festgelegten Speicherbereich des `BRAM` gelangen, bestehen aus `12 Bit` Realteil und `12 Bit` Imaginärteil. Da ein komplexer Eingang bei jedem Modul (vgl. Abbildung 3.2) im System vorgesehen ist, wird jede `32 Bit`-Zelle des Speichers in zwei gleiche *16 Bit* -Zellen aufgeteilt. Die Oberen `16` der `32 Bit` (vom `MSB` gesehen) enthalten den realen und die unteren 16 Bit den imaginären Wert eines Sensorsignals.Insgesamt `24` von `32 Bit` einer Speicherzelle werden mit aktiven Daten versorgt. Die restlichen `8 Bit` werden mit Nullen ausgefüllt und haben keinen direkten Einfluss auf die weiteren Berechnungen. Abbildung 3.7 zeigt die Datenaufteilung einer `32 Bit`-Speicherzelle im `BRAM`.

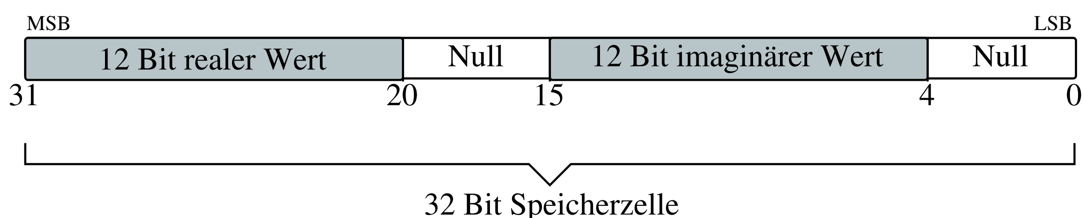


Abbildung 3.7: Darstellung der Sensordaten in einer 32 Bit BRAM-Speicherzelle

3.6 Speicherzugriff

Die Datenverarbeitung bei der vorliegenden Arbeit geht einer gewissen Befehlsabarbeitung bzw. Vorgehensweise nach, um zum einen die Berechnungen fehlerfrei durchzuführen und zum anderen Signalkonflikte beim Lese- und Schreibvorgang zu vermeiden.

3.6.1 Lesevorgang

Wenn ein Modul im System durch `Modul_Ctrl` aktiviert ist, wird automatisch gezählt und auf die fallende Taktflanke geachtet. Alle durchgeführten Aktionen oder Berechnungen werden durch die negative Taktflanke ausgelöst. Bevor die Daten verarbeitet werden, müssen sie aus dem Speicher geholt werden. Wie in Abbildung 3.8 dargestellt ist, wird bei dem Lesevorgang die Readleitung (`RE`) aktiviert und die Zieladresse (`ADDR`) angelegt. Nach einer Verzögerungszeit reagiert der `BRAM` auf die aktivierte Leitung. Bei der nächsten fallenden Flanke wird die Leseaktion (Readaction) durchgeführt und die Adresse der nächsten zum lesenden Zelle berechnet. Eine Leseaktion kann nur einmal die Daten einer 32 Bit Speicherzelle abholen, also einmal einen realen- und imaginären Wert. Soll zum Beispiel eine komplexe Addition zweier Werte berechnet werden, werden zwei Leseaktionen benötigt. Weitere Befehle die in den nächsten Kapiteln beschrieben werden, finden bei der negativen Taktflanke statt.

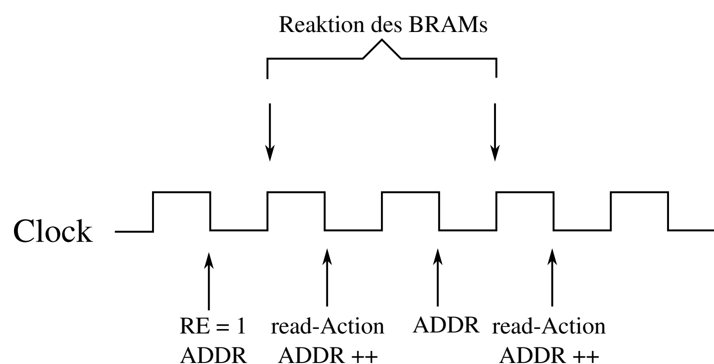


Abbildung 3.8: Lesevorgang beim Speicherzugriff

3.6.2 Schreibvorgang

Im Gegensatz zum Lesevorgang, werden beim Schreibvorgang die drei wesentlichen Befehle, Aktivierung der Schreibleitung `WE`, das Anlegen der Adresse `ADDR` die Durchführung der Schreibaktion, innerhalb derselben fallenden Taktflanke ausgeführt. Sobald die Schreibleitung `WE` und die schreibende Adresszelle `ADDR` aktiviert und angelegt sind, wird die Schreibaktion durchgeführt. Es ist zu beachten, dass bei dem Schreibvorgang die Readleitung `RE`, die vorher für das Lesen aktiviert wurde auf `low` gesetzt wird. Ansonsten werden weder ein Schreib- noch ein Lesevorgang stattfinden. Dieser Zustand kann durch einen Signalkonflikt ausgelöst werden. Eine Zusammenfassung der wichtigen Aktionen des Schreibvorgangs wird in Abbildung 3.9 erläutert.

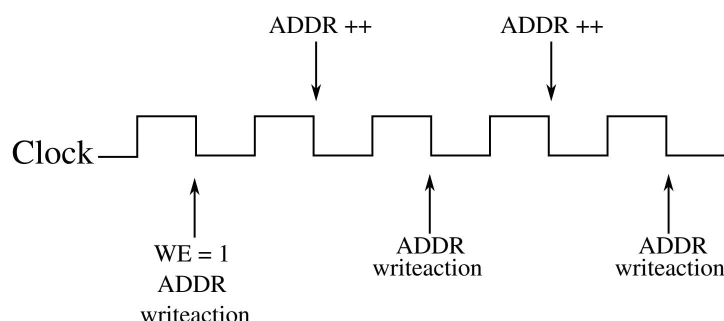


Abbildung 3.9: Schreibvorgang beim Speicherzugriff

Sowohl beim Lese- als auch beim Schreibvorgang, wenn die Adresse der nächst zu lesenden oder schreibenden Zelle berechnet wird, speichert man vorläufig den Wert intern, bevor er weiter auf die Adressleitung angelegt wird. Mit der Aktion `ADDR++` wird eine interne Berechnung und Speicherung des nächsten Adresswertes ausgeführt.

3.7 Das Zahlenformat S1Q10

Für die weiteren Berechnungen in der vorliegenden Arbeit wird das Format S1Q10 für die jeweiligen Zahlenwerte angewandt. Die Bedeutung der Parameter und Zahlen in dem Format wird im Folgenden erläutert:

- das S steht für das Zahlenvorzeichen und kann eine der Binärzustände annehmen. 1 für negative Zahlenwerte, 0 für positive Zahlenwerte
- Die `1` bedeutet bei der Schreibweise, dass nur eine Vorkommastelle erlaubt ist
- `10` : 10 Nachkommastellen sind hier vorgesehen

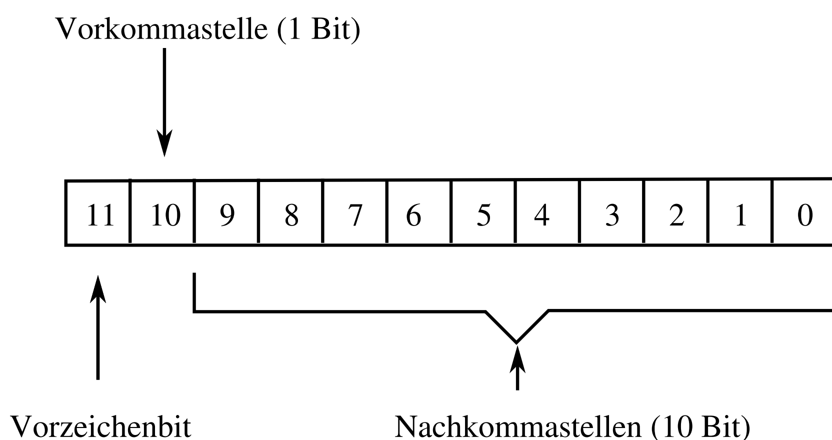


Abbildung 3.10: Das 12 Bit-Festkommazahlenformat S1Q10

Für die Zahlendarstellung und dessen Speicherbedarf im RAM sind für jeden Zahlenwert 12 Bit nötig. Das virtuelles Komma steht hinter dem Bit 10.

3.8 Hauptfunktionen

In diesem Abschnitt werden Hauptfunktionen schematisch beschrieben, die während der Aktivierung eines Moduls mehrfach aufgerufen wurden. Ein separates Modul `operation.vhdl` wurde zum Testen der einzelnen arithmetischen Grundoperationen, wie die Ermittlung des Mittelwertes, die Addition oder Subtraktion und die Multiplikation zweier Dualzahlen in dem festgelegten Zahlenformat S1Q10 implementiert. Das Modul hat an seinen Eingängen zwei separate `signed Input a[11:0]` und `b[11:0]` derselben Wortbreite 12 Bit und stellt an seinen Ausgängen die Resultate der einzelnen Grundoperationen dar. Dafür wurden als Output-Schnittstellen `add[11:0], diff[11:0], mult[11:0]` sowie `mean[11:0]` der Wortbreiten 12 Bit und demselben Datentype `signed` vorgesehen. Parallel dazu wurde in Matlab ein kleines Skript geschrieben für den späteren Vergleich der Simulationsergebnisse des Moduls und den Resultaten aus den Matlab-Berechnungen. Man konnte feststellen dass die einzelnen Grundoperationen mit Matlab übereinstimmen. Simulationsergebnisse, der Code des Moduls sowie das Matlab-Skript befinden sich im Anhang der Abschlussarbeit.

3.8.1 Ermittlung des Mittelwertes in Hardware

Wie in den Grundlagen erwähnt, wird im Modul Interpolation lediglich der Durchschnitt zweier sukzessiver Werten horizontal und vertikal ermittelt.
Hierfür wird eine Funktion in einer zuvor deklarierten `Package Functions_PKG` definiert. Die Funktion `calcInterpol(arg1, arg2)`, bekommt zwei 12 Bit Werte desselben Typs: `signed(data_width -1 downto 0)`, erweitert sie auf 13 Bit anhand der Funktion `resize()` der Bibliothek `Numeric_std` und ermittelt den Mittelwert. In Hardware ist

die Berechnung des Mittelwertes zweier Zahlen durch eine einfache Rechtsverschiebung (`Right bitshift`) dessen Summe durchgeführt. Aus dem 13 Bit Additionsergebnis, werden ab `MSB`, die ersten 12 Bit als Ergebnis betrachtet. Das `LSB` wird weggelassen, siehe Abbildung 3.11.

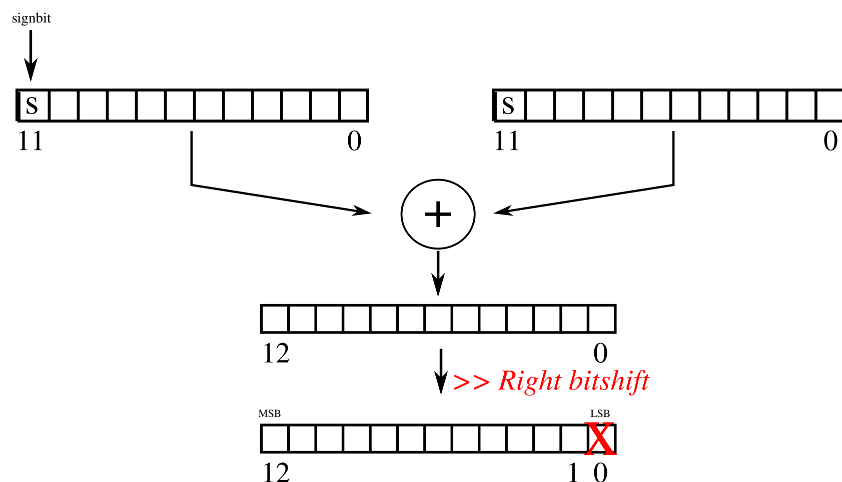


Abbildung 3.11: Berechnung des Mittelwertes zweier 12 Bit Festkommazahlen

3.8.2 Die Addition und Subtraktion zweier Festkommazahlen S1Q10 in Hardware

Die Bildung eines Zellenergebnisses bei der Matrixmultiplikation resultiert aus sequentieller Additionen von Teilprodukten der beteiligten Matrixzellen. Bei der vorliegenden Abschlussarbeit wird hauptsächlich mit dem zuvor dargestellten Zahlenformat S1Q10 berechnet, ein Format das es erlaubt, definierte Zahlengröße darzustellen. Um Überlaufe sowie Unterlaufe bei der Berechnung zu behandeln, werden Funktionen wie `add()` und `subtrac()` für die Addition bzw. die Subtraktion zweier Dualzahlen im `Package Operate_PKG` implementiert. Sowohl bei der Addition als auch der Subtraktion werden alle möglich auftretende Fälle untersucht, nämlich wie die Vorzeichen der zu addierten oder subtrahierten Dualzahlen stehen. Nachdem die Operation mittels der Funktionspackage `resize()` durchgeführt wurde, wird anhand `if,elseif, else` -Abfrage das Resultat der Operation zurückgegeben. Der in Abbildung 3.12 dargestellte Programmablauf zeigt Beispielweise wie die Addition zweier Dual-`S1Q10` bei der vorliegenden Entwicklung zu programmieren ist.

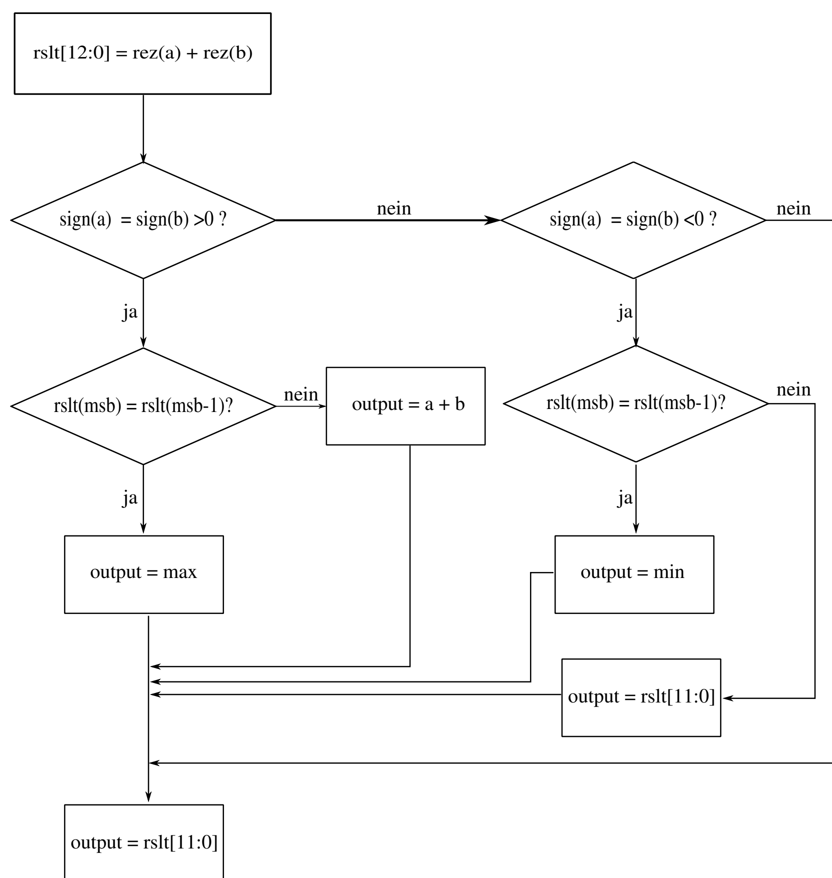


Abbildung 3.12: Addition zweier Dualzahlen in S1Q10-Format

3.8.3 Das Produkt zweier Festkommazahlen S1Q10 in Hardware

Wie in Abbildung 3.13 zu entnehmen ist, wird nach dem Produkt zweier in `S1Q10`-Format dargestellten Dualzahlen, das Resultat zunächst in einer Variable des Zahlenformats und Wortbreite `SS2Q20` bzw. 24 Bit gespeichert. Das endgültige Ergebnis ist die 12 Bit-Kombination aus `[21:10]`, wobei das Bit `21` das Vorzeichen des Resultats enthält, es resultiert aus einer einfachen `Kopie` des Bit `23-` in `21`. Die Bit-Reihenfolge aus `[20:10]` ist der Betrag und das virtuelle Komma ist nach der Stelle `19` vorzusehen. Die hier implementierte Vorgehensweise wird hauptsächlich ausgewählt weil die Twiddle-Faktoren, die bei der Multiplikation beteiligt werden, alle im Intervall]-1, 1] liegen. Das heißt, weder Überläufe noch Unterläufe werden bei der gewählten arithmetischen Operation auftreten. Code-mäßig reicht eine einfache Concatenation mit dem Operator (&) (Verknüpfen) des Bit 23 an den restlichen 11 Bit [20:10] um das richtige Resultat des Produktes zu erhalten

```
Output := pdc(23)&pdc(20 downto 10);
```

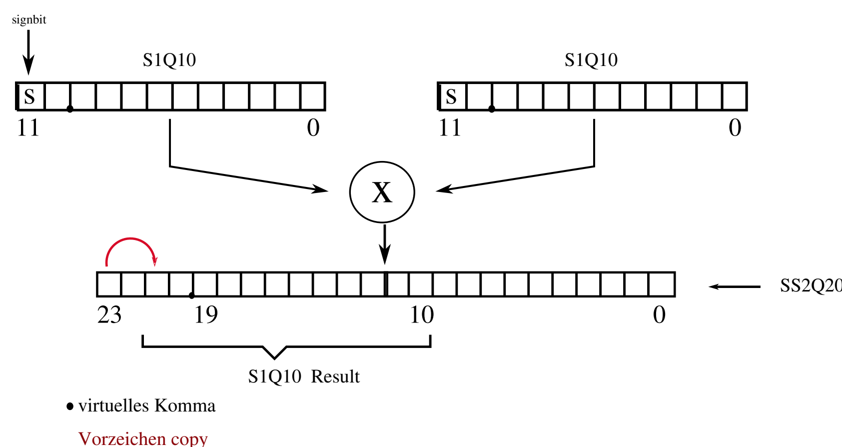


Abbildung 3.13: Resultat des Produktes zweier Dualzahlen in S1Q10-Format

3.9 Das Entwicklungsverfahren in VHDL

Für die Zielergebnisse wurden bei jedem Modul drei `Process()` entworfen. Sie reagieren auf bestimmte externe Signaländerungen `CLK, nRESET und modlctrl`, kommunizieren über weitere externe Signale wie `ctrl_op` miteinander und laufen asynchron zueinander. Ein `HIGHLEVEL` Zustand vom Signal `ctrl_op` hat als Folge die Datenbearbeitung im Modul zu unterbrechen. Dieser ist dann nur zu erwarten, wenn die Bearbeitung innerhalb eines Moduls oder Komponente zu Ende ist.
Die ersten zwei Prozesse sind dafür da, die Berechnungen der Zielergebnisse zu steuern.

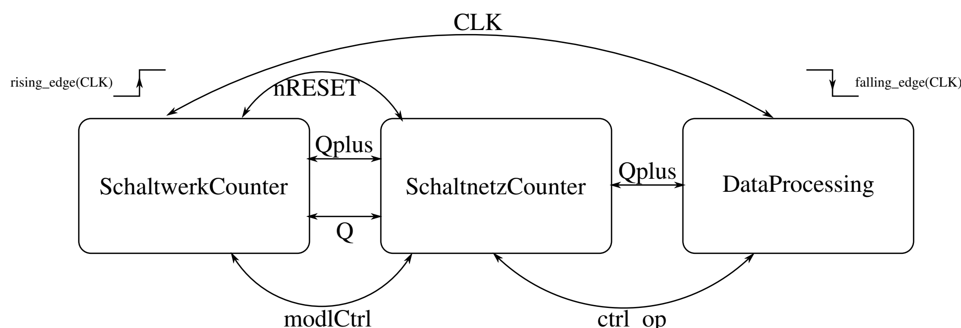


Abbildung 3.14: Aufteilung der Signalverarbeitung anhand Dreiprozessdarstellung

Wie in Abbildung 3.14 schematisiert, ist der erste `SchaltwerkCounter`-Process dafür da, den aktuellen Wert des Zählers bei positiver steigenden Taktflanke im lokalen Signal `Q` zu speichern und bei aktivem negativen `nRESET` also auf den `6 Bit` Wortbreite Null zurückzusetzen. Der Prozess ist asynchron implementiert.
Das Schaltnetz `chaltnetzCounter"` wird im Laufe der Datenverarbeitung immer abfragen, ob das Signal `Q` den maximalen Zählwert erreicht hat. Ist das der Fall, wird `Qplus`

an einem Zählzustand (`QCTN1`) zurückgesetzt. Dieser entspricht im `DataProcessing` dem Beginn der Datenverarbeitung (hier werden alle Signale und Variablen ihre Werte beibehalten). Ist die Abfrage nicht erfüllt, wird im Signal `Qplus` der nächste Zählwert gespeichert. Die weiteren Bearbeitungsschritte im `Process DataProcessing` werden in den nächsten Kapiteln ausführlich behandelt. In diesem `Process` finden alle wichtigen Operationsschritte zum erreichen der Zielergebnisse statt.

4 Interpolation

Vor der Modulimplementierung in VHDL wird zunächst mittels Matlabfunktion `interp2()`, welche die Interpolation eines Eingangsarrays berechnet, ein Test durchgeführt. Der hier durchgeführte Test, hilft dem Entwickler, seinen Algorithmus zu verstehen und besser zu gestalten. Bei dem Test wurde ein Eingangsdaten-Arrays (8×8) mit Zufallswerten ausgefüllt. Matlab `-interp2()` nimmt in ihrem Argument ein Daten-Array und bildet sequentiell die jeweiligen Mittelwerte zwischen zwei hintereinander liegenden Rohwerte (sukzessive Messdaten). Dabei werden sowohl die Zeilen- als auch die Spaltenanzahl, also die Dimension der Eingangsmatrix erweitert. Aus A (8×8) wird die interpolierte A' (15×15). Die folgende Abbildung 4.1 der Testergebnisse illustriert deutlich die Rechenschritte des Algorithmus.

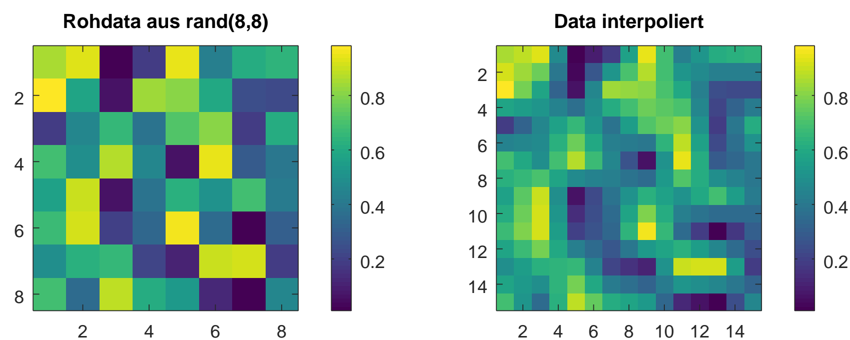


Abbildung 4.1: Interpolationstest aus (8×8) zu (15×15) mittels Matlab `interp2()`

4.1 Modularchitektur

Anhand des obigen Beispiels und durch viele Überlegungen lässt sich die Implementierung des ersten Moduls des Systementwurfs in Abbildung 3.2 (`Interpolation`) in VHDL bauen. Um die Berechnungen zu vereinfachen und somit eine ausführliche Vorgehensweise nachverfolgen, wird das Modul `Interpolation` als Toplevel von zwei weiteren Untermodulen oder Komponenten betrachtet. Die hier ausgewählte Vorgehensweise soll es ermöglichen, die Interpolation zeilenweise (horizontale Interpolation, `ROWINT`) und spaltenweise (vertikale Interpolation, `COLINT`) zu ermitteln. Die folgende Abbildung 4.2 beschreibt graphisch die innere Architektur der `Interpolation`. `ROWINT` und `COLINT` bilden die zwei Komponenten des Moduls und sollen für die weiteren Berechnungen sequentiell arbeiten. Die Aktivierung der `Interpolation` über das `Modul_Ctrl` (Abb. 3.2) schaltet automatisch ihre Komponente `ROWINT` ein, die entlang jeder Zeile der Eingangsmatrix den Mittelwert zwischen zwei sukzessiven Messdaten berechnet. Sind die 8 Zeilen

der Eingangsmatrix fertig bearbeitet, wird das `ready_Signal`, das hier als Kommunikationsleitung für die beiden Komponenten dient, auf `HIGH` gesetzt. Der `HIGH` Zustand des Signals `ready` wird automatisch die Berechnungen im `ROWINT` unterbrechen und aktiviert zeitgleich die Komponente `COLINT`. Somit ist die Berechnung der vertikalen Interpolation gestartet.

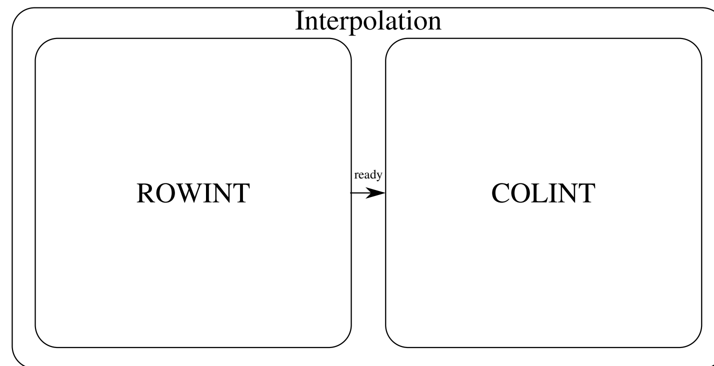


Abbildung 4.2: Architektur des Moduls `Interpolation`

4.1.1 Die Komponente `ROWINT` (horizontale Interpolation)

Die wesentliche Aufgabe der Komponente `ROWINT` ist die horizontale Interpolation, die Berechnung der Mittelwerte zweier sukzessiven Messdaten entlang jeder Zeile der (8×8) Eingangsmatrix. Hierfür werden zunächst die Sensordaten mittels des Aufbaus in Abbildung 3.1 in den BRAM geschrieben. $(64 \times 32Bit)$ komplexe Sensordaten liegen also als Rohdaten im Speicherbereich [0...63], siehe Abbildung 4.3.

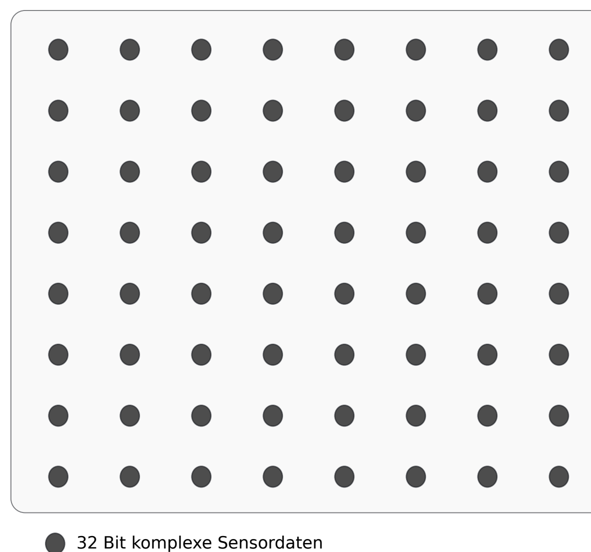


Abbildung 4.3: Komplexe Sensordaten (Eingangsmatrix (8×8))

Die Ein- und Ausgangssignale der Komponente `ROWINT` entsprechen denen des Toplevel-Moduls (`Interpolation`) bis auf das `ready` Signal, das die zwei internen Komponenten verbindet.
Um Signalkonflikte beim Schreib- und Lesevorgang zu vermeiden wird intern einen Multiplexer eingesetzt, welcher über das interne Signal `ready` die entsprechenden Eingänge bzw. Ausgänge der Komponente auswählt, siehe Abbildung 4.4. Bei einem `LOW` Zustand des `ready` Signals werden lediglich die Schnittstellen der Komponente `ROWINT` betrachtet , bei dem `HIGH` Zustand werden die von `COLINT` selektiert.

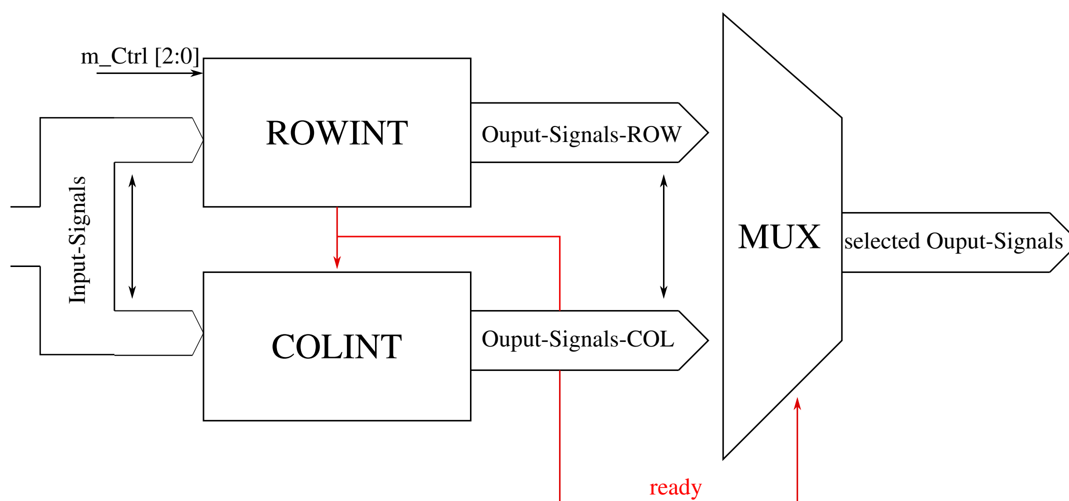


Abbildung 4.4: Auswahl der Ausgangsschnittstellen anhand eines internen MUX. Hiermit werden Signalkonflikte vermieden

Implementierung

Um einen Mittelwert zu bilden, sind mindestens zwei Sensordaten nötig. In einer Rohdatenzeile stehen gemäß Abbildung 4.3 acht komplexe Sensordaten zur Verfügung, ein Cosinuswert und ein Sinuswert, die hier als Real- beziehungsweise Imaginärteil betrachtet werden. Die einzelnen Berechnungsschritte und Befehlsausführungen laufen teilweise sequentiell und finden immer bei der negativen Taktflanke (`falling_edge(CLK)`) statt. Zunächst werden alle interne Signale und Variable initialisiert. In unserem jetzigen Fall (Komponente `ROWINT`), bekommen die Read- und `Writeleitung` den `"LOWLEVEL"`Wert die logische '0'. Read_Adress- (`RE_ADDR`) und Write_Adress(`WR_ADDR`) Leitungen werden an den Adressen 0 beziehungsweise 64 positioniert. Nach der Initialisierung, befindet man sich unmittelbar im Lesevorgang, bei dem die `Readleitung` (`RE`) auf `"HIGHLEVEL"`(logische '1') gesetzt wird, anschließend wird die Adresse (`RE_ADDR`) angelegt. Die beiden Befehle finden bei derselben fallenden Taktflanke statt.
Bei der nächsten fallenden Taktflanke können die ersten komplexen Sensordaten aus dem Speicher gelesen und vorläufig separat in Real- und Imaginärteil in einem Zwischenspeicher gespeichert werden, anschließend wird die nächste zu lesende Adresse um 1 erhöht

(`RE_ADDR ++`) und temporär gemerkt. Eine halbe Taktperiode wird gebraucht, um den zweiten Lesevorgang zu starten und die nächsten komplexen Eingangsdaten zu lesen.
Bei dem Zeitpunkt stehen $(4 \times 12Bit)$ separate Sensordaten($(2 \times 12Bit)$ Cosinuswerte oder Realteile und $(2 \times 12Bit)$ Sinuswerte oder Imaginärteile) zur Verfügung, um die komplexen interpolierten Daten zu berechnen. Bei der fallenden Taktflanke wird die Readleitung (`RE`) zurückgesetzt anschließend erfolgt die sequentielle Befehlsabarbeitung,

- Berechnung der neuen Adresse (für den nächsten Lesevorgang)
- Berechnung des imaginären interpolierten Wertes
- Berechnung des realen interpolierten Wertes
- Writeleitung (`WE`) aktivieren (`HIGHLEVEL`)
- Anlegender Schreib-Adresse (`WR_ADDR`) für den Schreibvorgang
- Schreiben der ersten Ergebnisse der Interpolation in das BRAM durch die `Package-Function writeData()`

Den Schreibvorgang in den Adressbereich der Interpolation kann man sich so vorstellen, dass die ersten gelesenen Sensordaten nach der Berechnungen des interpolierten Wertes in das BRAM geschrieben werden. Auf die folgende Adresse wird die Interpolation geschrieben, darauf das Datum des zweiten Lesevorgangs. Insgesamt wird dreimal geschrieben. Abbildung 4.5 zeigt die Datenreihenfolge im Speicherbaustein BRAM. Die hier

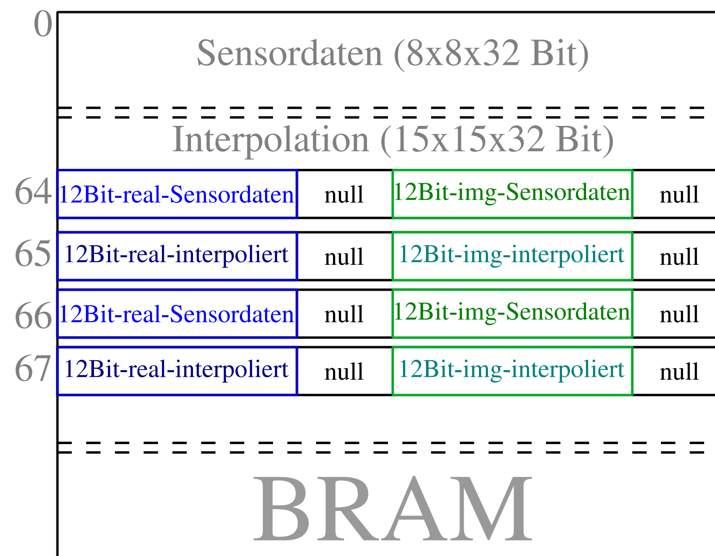


Abbildung 4.5: Datenreihenfolge im BRAM-Speicher / Speicherbereich für die Interpolation

oben beschriebenen Berechnungsschritte werden entlang der ersten Zeilendaten fortgesetzt und wiederholen sich in den restlichen sieben Zeilen der Eingangsmatrix. Damit die

Ergebnisse der Interpolation denen der Matlab-Funktion `interp2()` ähneln, wurde der Algorithmus so entwickelt, dass immer nach einer Zeilenberechnung, $(15 \times 32Bit)$ Speicherzellen frei gelassen werden, die später von der Komponente `COLINT` berücksichtigt und mit aktiven Werten ausgefüllt werden. Sobald die Read-Adresse `RE_ADDR` die Speicherzelle `63` erreicht, wird das `ready`-Signal aktiviert und die Komponente `ROWINT` wird verlassen, was gleichzeitig die Aktivierung der Komponente `COLINT` bedeutet. Der Programmablauf in Abbildung **??** zeigt die wichtigen Rechenschritte des implementierten Algorithmus.

4.1.2 Die Komponente COLINT (vertikale Interpolation)

Wie jede Komponente oder Modul im System, werden die Ein- und Ausgänge der Komponente `COLINT` direkt mit dem `BRAM`-Baustein verdrahtet bis auf das `ready`-Signal, das `ROWINT` mit `COLINT` verbindet. Somit werden die Lese- und Schreibvorgänge unmittelbar durchgeführt.
Der `high`-Zustand des Eingangssignals `ready` von `COLINT` startet die weiteren Berechnungsschritte des Interpolation-Algorithmus. Für die Berechnung der Komponente `COLINT`, werden hauptsächlich die $(7 \times 15 \times 32Bit)$ Speicherlücken betrachtet, die vorhin in `ROWINT` erzeugt wurden. Analog zu `ROWINT`, werden hier vor Beginn der Berechnung die wichtigen Adressleitungen und Variablen initialisiert.

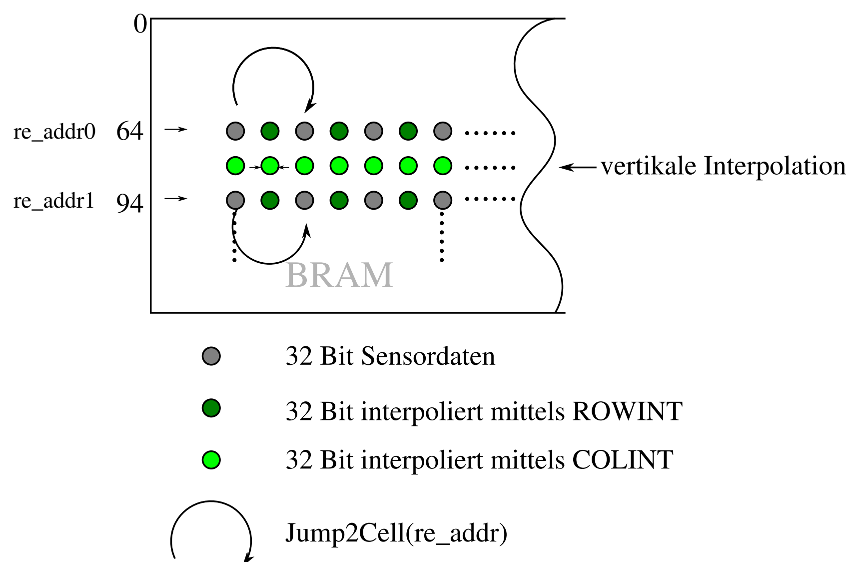


Abbildung 4.7: Graphische Darstellung der Rechenschritte in `COLINT`

Wie angedeutet in Abbildung 4.7, werden bei der Implementierung für den Lesevorgang zwei Adressleitungen (Read_Address) als Variable deklariert, `RE_ADDR0` und `RE_ADDR1`. Sie zeigen von Beginn an und bei aktivem `nRESET` auf die Adresse `64` bzw. `94`.

Für die Berechnung der ersten Ergebnisse der vertikalen Interpolation, werden zunächst die 32 Bit aus der ersten Adresse (64) gelesen, in Real-und Imaginärteil zerlegt und vorläufig gespeichert, dann wird die Variable `RE_ADDR1` an der Adressleitung angelegt um die darunter liegenden 32 Bit aus dem Speicher zu holen. Diese Lesevorgänge finden nur bei aktiver `Readleitung (RE = 1)` statt.
Sobald zwei Lesevorgänge durchgeführt wurden, wird das Unterprogramm für die Berechnung der Interpolation aufgerufen. Die ersten Ergebnisse sind hiermit ermittelt und für weitere Berechnungen vorläufig gespeichert.
Anschließend wird innerhalb derselben fallenden Taktflanke die Readleitung wieder zurückgesetzt und die Adresse für den nächsten Lesevorgang erneut berechnet. Hierfür wurde eine einfache Funktion implementiert `jump2Cell(addr)`, die als Argument eine Adresse annimmt und diese um `2` inkrementiert. Die Funktion ermöglicht einen einfachen Sprung um 2 Speicherzellen. Sie gibt als Rückgabewert eine Adresse der Wortbreite `10` zurück.
Die Write_Adresse `wr_addr`, die bei Initialisierung dem Adresswert 79 zugewiesen wurde, wird an der Adressleitung angelegt und sequentiell erfolgen die Aktivierung der Writeleitung (`WE = 1`) und das Schreiben der Ergebnisse in das`BRAM`. Die nächste fallende Taktflanke ist der Zeitpunkt, wo die Writeleitung wieder zurückgesetzt wird. Die zuvor berechnete `RE_ADDR0` wird erneut an der Adressleitung angelegt und anschließend erfolgt die Aktivierung der Readleitung. Die nächsten 32 Bit können gelesen werden und wie zuvor separat in Real- und Imaginärteil zerlegt und gespeichert werden. Identisch werden die Daten aus der Adresse `RE_ADDR1` genauso verarbeitet. Dann folgt die Ermittlung der Ergebnisse der Interpolation. Zwei Speicherzellen wird gesprungen um die hier ermittelte Ergebnisse zu schreiben. Aus den beiden Interpolation-Ergebnissen der vier Lesevorgänge wird der Mittelwert berechnet und anschließend in das `BRAM` geschrieben. Abbildung 4.7 illustriert die Rechenschritte.
Das Verfahren soll solange durchgeführt werden, bis die hell grüne Zeile $(15 \times 32 Bit)$ mit Daten ausgefüllt werden. Es wird anschließend abgefragt, ob die `RE_ADDR1` die Speicherzelle `288` erreicht hat. Ist das NICHT der Fall, wird mittels der Funktion
`jump2NextRow_ADDR(addr)` um $(15 \times 32 Bit)$ Speicherzellen gesprungen und nächsten Zeilen werden zum Lesen und Schreiben der Daten für die restlichen Berechnungen selektiert. Ist aber die Abfrage erfüllt, wird dem Ausgangssignal `modul_ready` der Wert `1` zugewiesen. Die Interpolation ist hiermit komplett berechnet und die Ergebnisse endgültig in den Speicherbaustein geschrieben. Es werden für die Komponente `COLINT 231 Takte` nötigt, um die Interpolation wie `Matlab-interp2()` zu berechnen.
Die Interpolation wurde hauptsächlich nur mit den komplexen Sensorrohdaten berechnet. Die Zwischenergebnisse aus der Komponente `ROWINT` wurden hier in `COLINT` nicht einmal angefasst, um die weiteren Daten zu ermitteln. Das Verfahren in der zweiten Komponente `(COLINT)` hat als Vorteil die Reduzierung der Anzahl der Takten bei der Datenverarbeitung. Mittels der Funktion `jump2Cell(addr)`, ist das dann nicht mehr nötig jede 32 Bit Daten zu lesen um Ergebnisse daraus zu ermitteln. Ein einfacher Sprung um zwei Zellen ermöglicht innerhalb `8 Takte` drei Werte zu berechnen und in das `BRAM` zu schreiben. Ein sequentielles Lesen jeder `32 Bit` bis zum Speichern der berechneten

Ergebnisse würde mehr Takte benötigen und somit eine langsame Bearbeitungsvorgehensweise bedeuten.

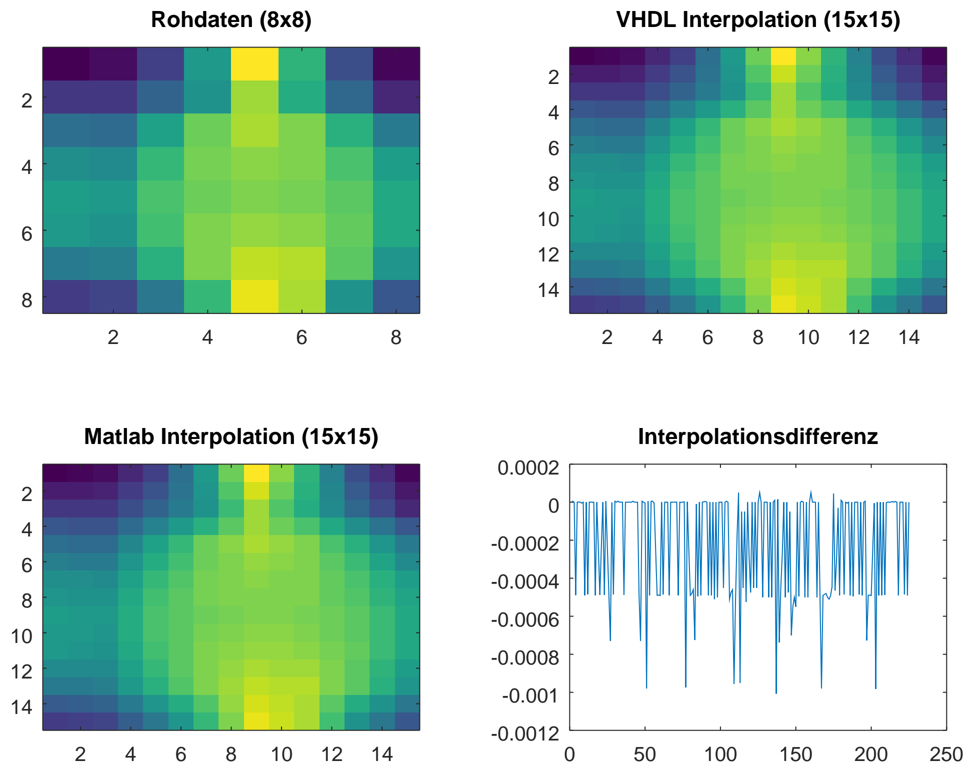


Abbildung 4.8: Interpolation VHDL und Matlab verglichen

5 Die Zweidimensionale diskrete Fouriertransformation (2D-DFT)

Das 2D-DFT-Modul ist die VHDL-Implementierung des in Gleichung 5.1 dargestellten Algorithmus.

$$X = W_{\mathrm{M}} \cdot x \cdot W_{\mathrm{N}} \tag{5.1}$$

Um die VHDL-Implementierung zu vereinfachen, wird die Gleichung 5.1 in zwei weitere Gleichungen zerlegt.

- `die 1D-DFT-Gleichung`,

  $$X_{\mathrm{1d}} = W_{\mathrm{M}} \cdot x \tag{5.2}$$

  welche die erste komplexe Matrizenmultiplikation zweier Matrizen derselben Dimension ($15 \times 15 \times 32Bit$) berechnet und als Ergebnis eine Datenmatrix ($15 \times 15 \times 32Bit$) ergibt, die hier als X_{1d} bezeichnet ist. Es werden also die Zeilen der Twiddle-Matrix (W_{M}) mit den Spalten der komplexen Eingangsmatrix (x) multipliziert.

- `die 2D-DFT-Gleichung`,

  $$X_{\mathrm{2d}} = X_{\mathrm{1d}} \cdot W_{\mathrm{N}} \tag{5.3}$$

  die anschließend das Produkt aus dem Ergebnis aus Gleichung 5.2 und derselben Twiddle-Matrix ($W_{\mathrm{M}} = W_{\mathrm{N}}$) bildet. Es ist festzustellen, dass die Parameter M und N bei den Gleichungen gleich groß sind, nämlich = 15.

Im VHDL-Entwurf werden dafür zwei Module implementiert:

- `1d_Mod` kümmert sich um die Implementierung der Gleichung 5.2
- `2d_Mod` gibt das Endergebnis der zweidimensionalen Diskreten Fouriertransformation laut Gleichung 5.3.

In Abbildung 5.1 lässt sich eine grobe innere Architektur des gesamten Topmoduls (2D-DFT) darstellen.

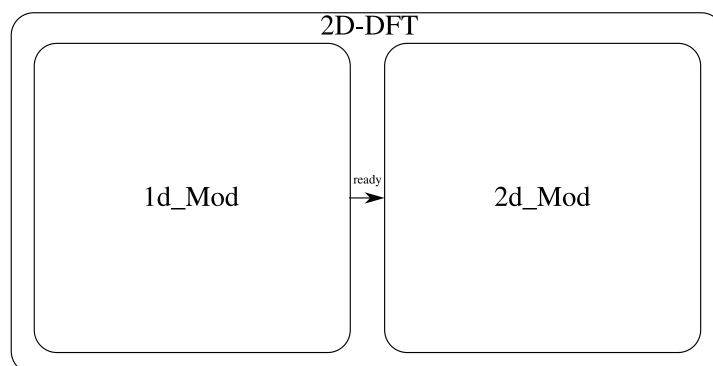


Abbildung 5.1: 2D-DFT Architektute

5.1 Bestimmung der Twiddle-Faktoren

Um die Faktoren der Twiddle-Matrix zu berechnen, wurde ein kleines Script mittels `Matlab`(alternativ: `Octave`) geschrieben. Das Script soll die Gleichung 5.4 als Funktion enthalten. Eine (15×15) Matrix mit komplexen Faktoren in der Darstellungsform ($\mathtt{a}+i\mathtt{b}$) wird berechnet.

$$TW_Matrix = exp((-\mathtt{i} \cdot 2 \cdot \pi \cdot [0:\mathtt{M}]' \cdot [0:\mathtt{M}])/\mathtt{M}) \tag{5.4}$$

wobei $\mathtt{M} = 15$ ist. Die Größe der Twiddle-Matrix muss mit der der Eingangsmatrix übereinstimmen.
Eine genauere Analyse der Twiddle-Matrix weist in ihrem realen Anteil zwei Achsensymmetrien auf, welche die Twiddel-Matrix in ihrem Realteil (Abbildung:5.2)) in vier gleich aufgeteilte Quadranten darstellt. Dieselben Eigenschaften lassen sich im imaginären Anteil TW-Matrix ablesen, nur da sind die Faktoren invertiert.
Aufgrund dieser Merkmale, werden für weitere Berechnungen des gesamten Algorithmus in Gleichung 5.1 nur die ersten Quadranten betrachtet. Also $(7 \times 7 \times 32Bit)$ komplexe Twiddle-Faktoren werden im RAM gespeichert.

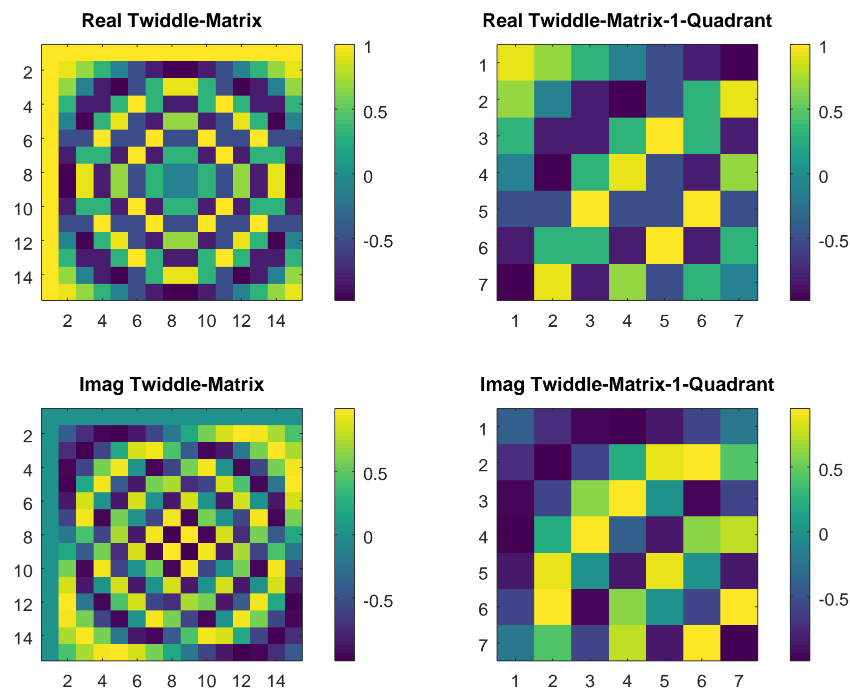


Abbildung 5.2: Darstellung der komplexen Twiddle-Matrix in Real- und Imaginärteile

5.2 Modularchitekture

5.3 1D-DFT

Ziel bei dem Modul ist, das erste Matrixprodukt (Gleichung: 5.2) der 2D-Diskreten-Fourier-Transformation zu implementieren. Um die Berechnungen zu vereinfachen und soweit wie möglich das Einsetzen des Multiplikationsoperators zu reduzieren, sind bei dem aktuellen `1D_DFT` zwei weiteren Komponenten vorgesehen, `addUnit`, für die Bildung der ersten Zeile des Matrixergebnisses und `operatingUnit`, die die restlichen Daten der Ausgangsmatrix ermittelt. Die Abbildung 5.3 zeigt, wie die erwähnten Komponenten miteinander kommunizieren. Sie werden sequentiell aktiviert, starten die jeweilige Befehlsabarbeitung und schreiben die erzielten Ergebnisse an eine bestimmte Speicherzelle des angeschlossenen `BRAM`.

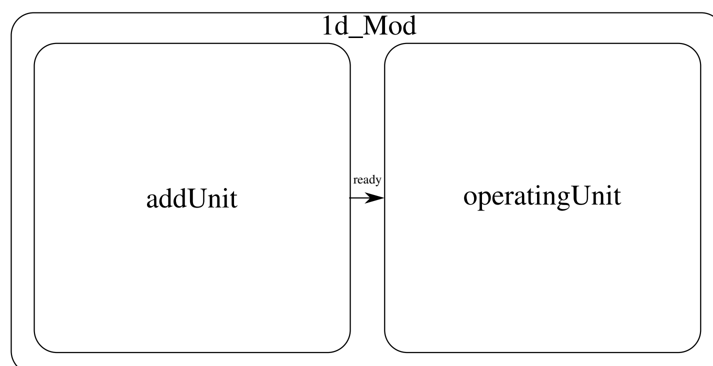


Abbildung 5.3: Verdrahtung der Komponente im Modul `1d_Mod`

5.3.1 Komponente addUnit

Ziel der jetzigen Komponente `addUnit` ist, die erste Zeile der 1D-DFT $(15 \times 15 \times 32Bit)$ komplexem Ausgangssignal zu bilden. Aufgrund der Konfiguration der Twiddle-Matrix, die in ihrer ersten Zeile beim Realteil nur aus Einsen und beim Imaginärteil aus Nullen besteht, lässt sich die Berechnung der einzelnen 15 Zellen der $(15 \times 15 \times 32Bit)$ komplexen Ausgangsmatrix ohne den arithmetischen Operator "Multiplikation"bilden. Jede der gebildeten 32 Bit Zellen der ersten Zeile der 1D-DFT ist eine sequentielle komplexe Addition der entsprechenden Spalten des Dateneingangs (komplexe Interpolation). Um die Berechnungsschritte spaltenweise zu realisieren, wurden unter anderem zwei wichtigen Funktionen implementiert:

- `jump2nextaddr(addr)`: soll es ermöglichen, nachdem ein Wert aus dem Speicher gelesen wird, zu der nächsten darunter liegenden Adresse derselben Spalte zu springen, die read-Aktion der 32 Bit wird ausgeführt, die in 12 Bit-Real und 12 Bit-Imag zerlegt wird, um anschließend die Addition der Real- und Imaginärteile auszuführen. Die Funktion nimmt in ihrem Argument einen Wert von Type `std-logic-vector(9 downto 0)` der Wortbreite 10 Bit an und gibt einen Wert desselben Typs zurück. Die Differenz zwischen der Adresse A (Funktionsargument) und der Adresse B (Funktionsrückgabewert) sind $(15 \times 32Bit)$ Speicherzellen.

- `jump2_1row(addr)`: Nachdem die Berechnung eines Spalteneingangs durchgeführt und die Ergebnisse mittels der `Writeleitung` (`WE`) und der angelegten `Writeaddress` in den Speicher geschrieben wurden, wird anschließend bei der nächsten negativen Taktflanke, anhand einer `if-Abfrage` geprüft, ob die letzte Zelle der letzten Spalte der Eingangsmatrix erreicht ist. Wenn `Ja` wird automatisch die `ready-Leitung` welche das Berechnungsende der Komponente signalisiert, aktiviert. Die Datenbearbeitung in der Komponente ist zu dem Zeitpunkt beendet. Die folgende und über die Ready-Leitung angeschlossene Komponente wird somit automatisch aktiviert und startet ihre Berechnung. Weitere Zellen des Speichers (`BRAM`) werden dann mit aktuellen Daten ausgefüllt. Wenn `Nein`, wird die nächste

Spalte für die Berechnung der weiteren 32 Bit Zellen, ausgewählt. Diese ermöglicht die Funktion `jump2_1row(ADDR)`. Wie hier dargestellt, ist in ihrem Argument eine Adresse (`ADDR`) desgleichen Typs wie bei der Funktion `jump2nextaddr(ADDR)`. Auch der Rückgabewert ist eine Adresse vom gleichen Typ. Bei der Funktion wird von der aktuellen Adresse $(209 \times 32Bit)$ abgezogen damit der Adresszeiger genau an der Stelle steht ab der berechnet wird.

Die jeweiligen Berechnungschritte und Befehlsabarbeitungen sind im folgenden Programablauf Abbildung 5.4 zu entnehmen. Jeder Zustand entspricht einer negativen Taktflanke `falling_edge(CLK)`. Von Zustand zu Zustand ist eine Taktperiode nötig. Um eine 32 Bit Ausgangsspeicherzelle zu bilden sind 31 Takte nötig. **465 Takte** werden benötigt, um die $(15 \times 32Bit)$ Speicherzelle mit aktuellen Werten zu schreiben.

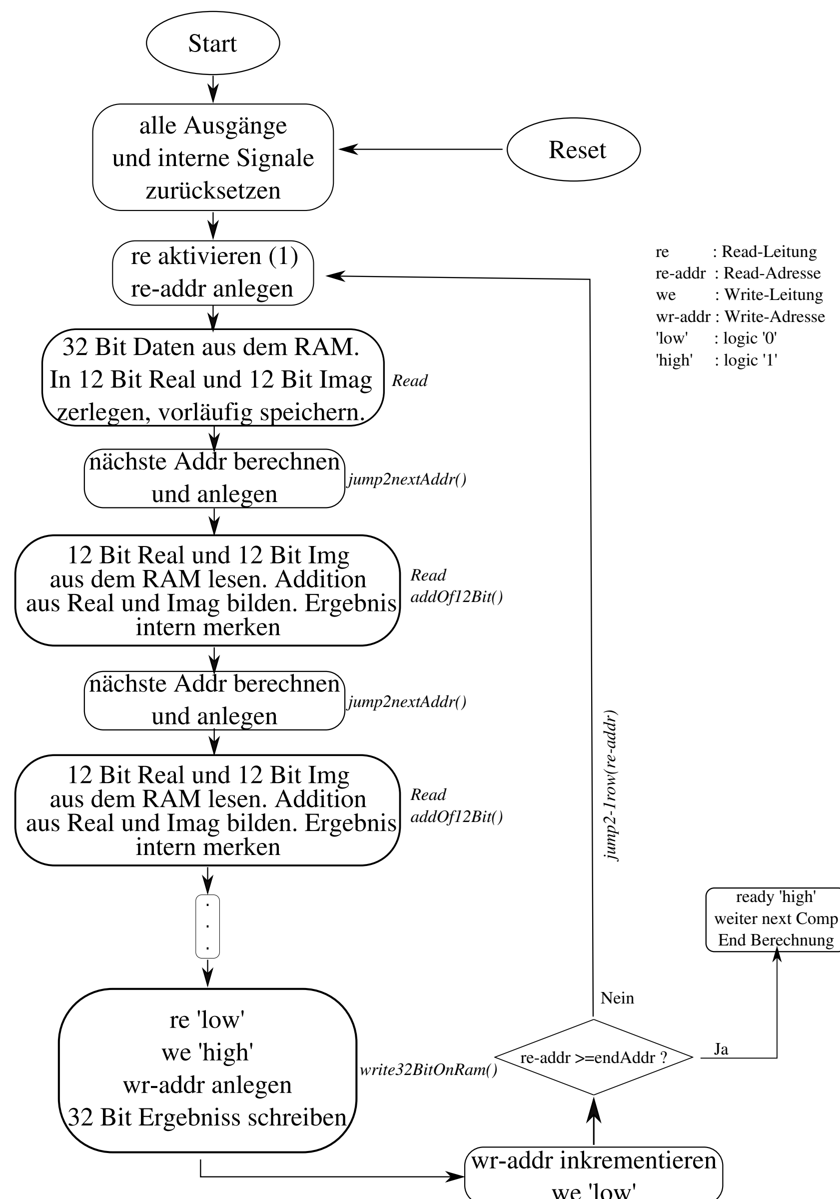


Abbildung 5.4: Programmablauf der Komponente addUnit.vhdl

5.3.2 Komponente OperatingUnit

`OperatingUnit` soll jetzt die restlichen $(210 \times 32Bit)$ Zellen der `1D-DFT` berechnen und die Ergebnisse im `BRAM` ab der Adresse `304` schreiben, da die ersten oberen $(15 \times 32Bit)$ ab Adresse `289` von der Komponente **`addUnit`** erfolgreich berechnet und geschrieben wurden. Für die Bildung der weiteren einzelnen `32 Bit` Zellen, wurde nach Analyse der Twiddle-Matrix, folgendes Verfahren angenommen.
Ausgehend von der Achsensymmetrie der `Twiddle-Matrix` ab Spalte `7`, stellt man bei-

spielsweise fest, dass die Werte aus den Adressen `79` und `274` mit demselben Twiddle-Faktor multipliziert werden und die darunter bzw. darüber liegenden Adresswerte von `79` bzw. `274` mit demselben Verfahren kalkuliert werden. Dies ermöglicht es, anstatt 14 Multiplikationen für die Bildung eines Zellenwertes durchzuführen, wird lediglich die Hälfte des Operators angewandt. Die Werte der gewählten Adresse werden hier zunächst zusammengefasst also addiert, bevor sie mit dem entsprechenden Twiddle-Faktor multipliziert werden. Die einzelnen Produktresultate werden addiert. das Prinzip soll den Chipflächenbedarf reduzieren und die Zielergebnisse werden somit schneller erreicht, weniger Takte werden für die Bildung eines Zellenwertes hierfür nötig. Abbildung 5.5 macht davon eine deutliche Illustration der eingesetzten Berechnungsvorgehensweise.

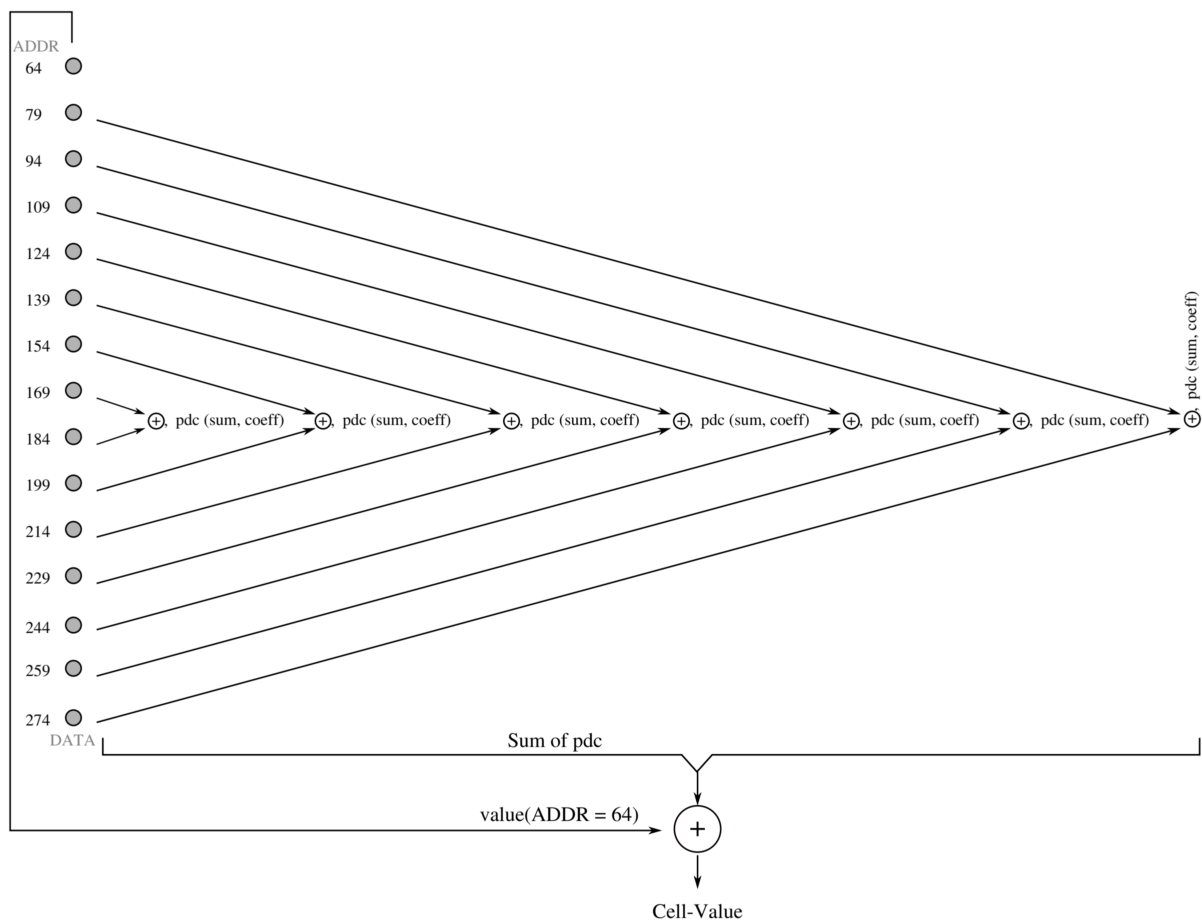


Abbildung 5.5: Zellbildung aus der Summe sequenzieller Produktteilen

Implementation

Nachdem die `Read`- und `WriteAdresse` (`79-274` bzw. `304`) sowie alle beteiligten Leitungen intern initialisiert wurden, wird in den Lesevorgang gesprungen. Diese Pha-

se entspricht der sequentiellen Befehlsabarbeitung der Aktivierung der `Readleitung`, das Anlegen der gewählten Adressen und das Lesen der `32 Bit` Datenzellen. Dieselben Schritte werden nochmals wiederholt, anschließend werden die Real- und Imaginärteile der einzelnen `32 Bit` addiert. Es folgen während der folgenden fallenden Taktflanke das Lesen des entsprechenden Twiddle-Faktors und die Berechnungen der nächst zu lesenden $(2 \times 32 Bit)$ Adressdaten anhand der Funktion `jump2nextaddr(ADDR)`. An dem Zeitpunkt wird das erste Produkt aus Datenwerte und Twiddle-Faktor gebildet. Man muss bedenken, dass es sich dabei um eine komplexe Multiplikation der Form $(a + ib) * (x + iy)$ handelt. Das Ergebnis ist ebenso komplex und lässt sich in dieser Form zusammenfassen $(ax - by) + i.(ay + bx)$. Das Resultat wird separat in Realteil- und Imaginärteil-Variable aufgeteilt und vorläufig gespeichert. Sie werden sequentiell zum nächsten Resultat des folgenden Lesevorgangs aufaddiert bis die gesamte Spalte der Eingangsmatrix abgearbeitet wird, somit stehen die ersten `32 Bit` Zellendaten aus der zweiten Zeile der Ausgangsmatrix der `1D-DFT`.
Nach der Bildung einer 32 Bit Zelle, werden alle Variablen zurückgesetzt, die nächste zu schreibende Zellenadresse wird mittels des Inkrementieren-Operators ermittelt und zu diesen Spaltendaten gesprungen. Die Bearbeitungsroutine wird erst verlassen, wenn alle 7 Zeilen der gespeicherten Twiddle-Matrix angesprochen wurden.
Aufgrund der Symmetrie der Twiddle-Matrix werden die berechneten realen Werte zu entsprechenden Speicherzellen kopiert, bei den imaginären Werte dagegen wird vor der Kopie, invertiert.

Programmablauf

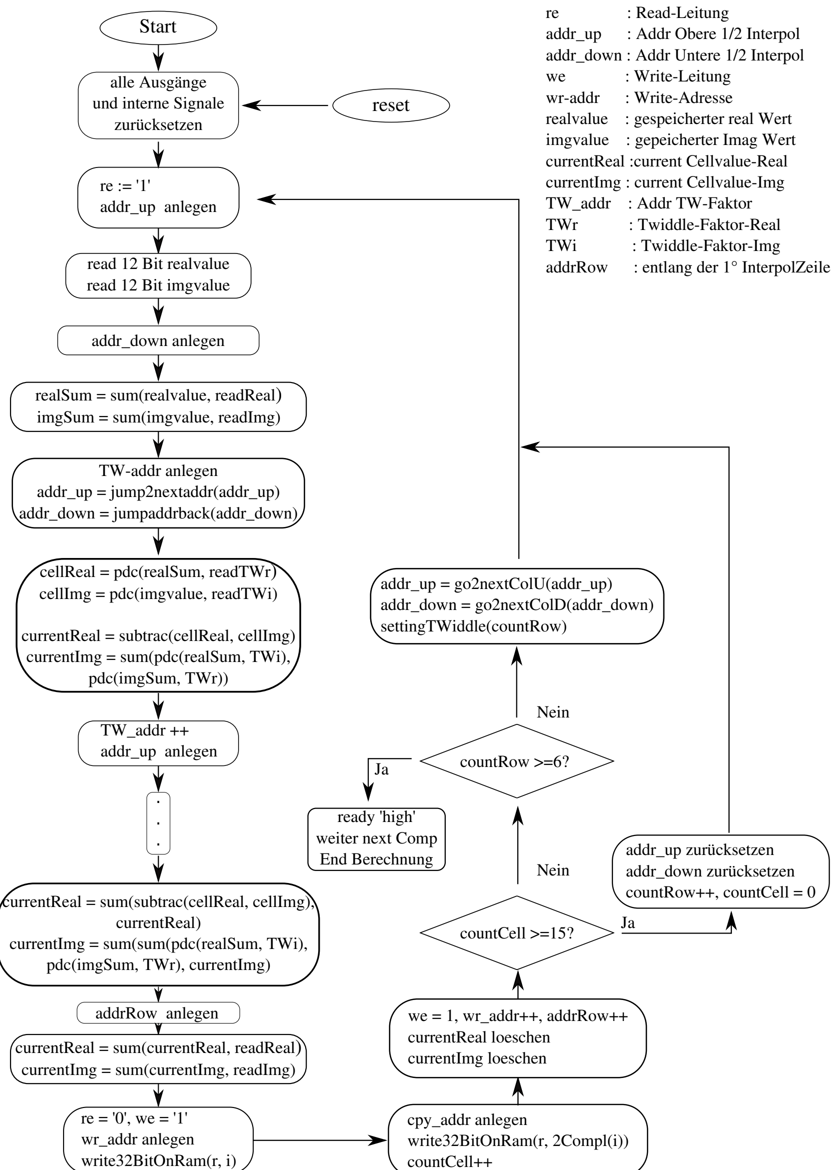


Abbildung 5.6: Rechenablauf der Komponente OperatingUnit

5.4 2D_DFT

Für die endgültige Ermittlung des Ergebnisses der `2D-DFT`, übernimmt das Modul `2d_Mod` die restlichen Berechnungen. Das Modul wird wie seine Vorgängen ebenso in Komponenten zerlegt. Die Komponente `COLDFT` die, aufgrund der Twiddle-Matrix Konfiguration die erste Spalte der `2D-DFT` bildet, und die Komponente `DFT2D` für die weiteren Zellen.

5.4.1 Komponente COLDFT

Mit `ColDFT` wird die Implementierung der `2D-DFT` gestartet. Die Komponente wird erst aktiviert, wenn die Berechnung der `1D-DFT` beendet ist. Ziel dieser Komponente ist die sequentielle Ermittlung der ersten Spaltenzellen der komplexen Matrixmultiplikation der `2D-DFT`. Der Grund der algorithmischen Vorgehensweise lässt sich aus der Konfiguration der Twiddle-Matrix entnehmen. Bei der `2D-DFT` geht es hauptsächlich um den zweiten Teil des Matrixproduktes der Gleichung 5.3. Der Reihe nach werden die Zeilen des komplexen Eingangs der `1D-DFT` mit den Spalten der komplexen Twiddle-Matrix multipliziert. Die hier ausgewählte Komponente bezieht sich nur auf die erste Spalte der Twiddle-Matrix, welche aus Einsen und Nullen in ihren Real- bzw. Imaginärteilen besteht und bildet die ersten Zellen der ersten Spalte durch Multiplikation der Eingangszeilen mit der ersten Spalte der Twiddle-Matrix. Eine Multiplikation wird hier nicht durchgeführt und die Berechnung des Ergebnisses entsteht aus den $(15 \times 32Bit)$ Zellenaddition der Zeilen der Eingangsmatrix. Es wird für jedes einzelne Ergebnis `15` mal gelesen, `14` mal addiert und `zweimal` geschrieben. Die erste Schreibaktion entspricht einer Übertragung des Rechenergebnisses in den RAM und die zweite Schreibaktion einer `Copypaste` dieses Ergebnisses an einer genau ausgewählter Stelle des Speichers gemäß der vertikalen Symmetrie der Twiddle-Matrix. Für die Berechnung wurden einfache Schritte sequentiell durchgeführt.

- Erstmal wird innerhalb einer fallenden Taktflanke die Readleitung aktiviert und Adresse angelegt
- An der nächsten fallenden Flanke werden `32 Bit` komplexe Eingangsdaten aus dem RAM gelesen, die in 12 Bit Real und Imag zerlegt werden. Dabei wird die Addr für die nächste Readaction durch eine einfache Inkrementierung berechnet
- Bei der dritten fallenden Flanke wird die eingangs berechnete Adresse wieder angelegt. Die zweite Readaktion wird durchgeführt, anschließend wird die ersten komplexe Addition für das Ergebnisziel ermittelt. Das Ergebnis wird intern vorläufig gespeichert

Die sequentielle Befehlsabarbeitung aus Adresse anlegen, Durchführung der Readaktion, Addition der aktuell gelesen Daten mit den vorläufig gespeicherten Daten wird solange fortgesetzt, bis das Ende einer ausgewählten Zeile erreicht ist. An dem Zeitpunkt ist die erste 32 Bit komplexe Wert der Matrixmultiplikation gebildet. Er wird mittels `writeData()` in den `BRAM` geschrieben. Wird die Write-Adresse (`619`) erreicht, erfolgt automatisch die Aktivierung des Ready-Signals und die Beendigung der Berechnungsaktionen innerhalb der Komponente. Sonst wird zu den nächsten Zeilendaten gesprungen und die Write-Adresse an die richtige Schreibstelle positioniert. **7 x 33 = 231 Takte** sind nötig für die gesamten Berechnungen.

5.4.2 Komponente DFT2D

Die Implementierung der Komponente `DFT2D` nutzt die vertikale Symmetrie der Twiddle-Matrix, um die Berechnungszeiten und damit die Taktanzahl zu reduzieren. Anstatt alle restlichen 14 Matrixspalten zu betrachten, werden nur die bereits im `BRAM` gespeicherten Koeffizienten genutzt. Eine wesentliche Entscheidung vor der Implementierung ist an welcher Speicherstelle die `Schreib-`und `Leseaktionen` zu positionieren sind. Da die ersten Spaltenzellen der `2D-DFT` mittels `COLDFT` mit aktiven Werte ausgefüllt sind, ist das von daher verständlich dass, die `Writeadresse` ab der Adressenstelle `515` starten soll, und die weiteren Rechnungsergebnisse in den Speicher zu übertragen.
Für die `Readaktion` werden dafür aufgrund der Symmetrie der Twiddle-Matrix zwei `Readadressen` vorgesehen. Der Vorteil der Vorgehensweise ist die Reduzierung der Multiplikationsoperators. Anstatt bei jedem zweiten Lesevorgang eine Multiplikation auszuführen, wird ein Produkt bei dem jetzigen Verfahren erst nach drei Lesevorgänge einmal durchgeführt.
Nachdem die wesentlichen Signalen und Variablen beim Zählzustand `QCTN0` initialisiert sind , benötigt man für die Berechnung des ersten Teilproduktes einer Speicherzelle insgesamt `6 Takte`, die sich in folgenden Bearbeitungsschritte aufteilen lassen,

- Angenommen, dass die wesentlichen Leitung und `Readadresse` aktiviert bzw. angelegt sind, startet man den Lesevorgang der ersten `32 Bit` Zellen ab der Adresse `addrfor (290)`. Da es sich hier um die Multiplikation aus Zeilen der `1D-DFT` und der Spalten der Twiddle-Matrix handelt, wird für die nächste Readaktion die Adresse `addrfor` um eine Speicherzelle ($1 \times 32Bit$) inkrementiert
- Anschließend zeigt die Adresse auf die Variable `addrback`, die den Adresswert der letzten Zellen der ersten Zeileneingang enthält. Eine Zusammenfassung aus den Real- und Imaginärteile der beiden Lesevorgänge mittels der `add()`-Function im Package `Functions-PKG`, ermöglicht es, einen komplexen Wert vorläufig zu speichern. Sequentiell wird der nächste Adresswert für die Variable `addrback` berechnet, eine Dekrementierung um eine Speicherzelle ist dafür nötig
- Der nächste Schritt entspricht dem Lesen des komplexen Twiddle-Faktors, nachdem die dafür initialisierte Adresse (`addrTW`) angelegt wurde. Es folgt mittels der `Package-Function mult()`, das Produkt aus komplexem Twiddle-Faktor und den zuvor gelesenen Eingangsdaten. Der daraus resultierende komplexe Wert wird vorläufig gespeichert und `6 Takte` später zu dem nächsten berechneten Wert nach derselben Vorgehensweise addiert.

Diese soll solange durchgeführt werden bis die aktuelle bearbeitende Eingangszeilen fertig ist. Abbildung 5.7 zeigt die wesentliche Berechnungsschritte für die Bildung einer Speicherzelle.

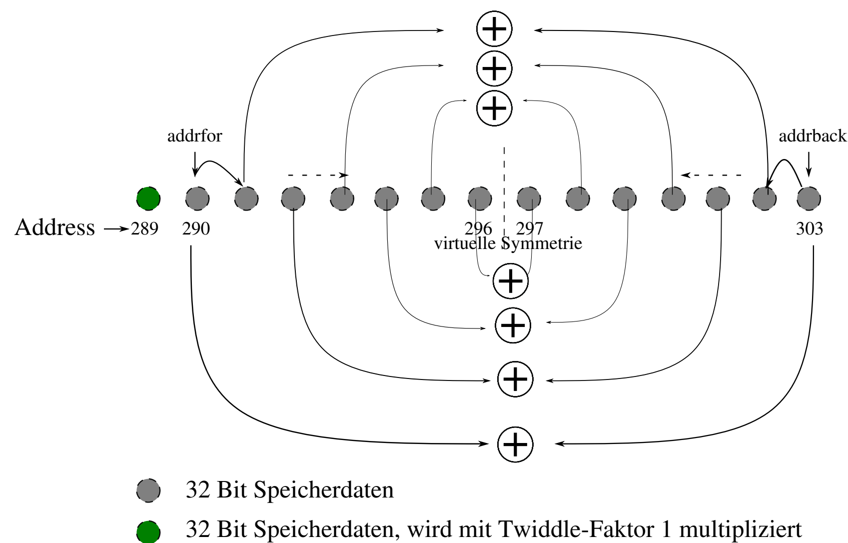


Abbildung 5.7: Berechnungsschritte für die Bildung einer Speicherzelle

Man muss auch darunter verstehen, dass nach jeder Summer wird automatisch in der nächsten fallenden Flanke mit dem entsprechen Twiddle-Faktor multipliziert. Anhand der vertikalen Symmetrie wird für eine Zellenbildung im `BRAM` 27 Mal gelesen inklusive die Readaktion der Twiddle-Faktoren, die in der Abbildung nicht dargestellt ist und zweimal geschrieben. Die erste Schreibaktion ist die sofortige Übertragung des Ergebnisses der sequentiellen Summe der Teilprodukten und die zweite geschieht durch die Nutzung der vertikalen Symmetrie. Für eine Zellenbildung sind `48 Takte` benötigt, da wo für eine Zeilenausgangsdaten 15 Speicherzellen nötig sind und unter der Berücksichtigung der horizontalen und vertikalen Symmetrie der Twiddle-Matrix, kommt man auf `5040 Takte` insgesamt . Abbildung zeigt den Programmablauf des implementierten Algorithmus in Komponente `DFT2D`.

6 2D-Digitalfilter

Wie die 2D-DFT, werden bei der Berechnung im Modul 2D-Filter, die Koeffizienten aus einem festgelegten Speicherbereich des Block-RAM ADDR = [788-907] ausgelesen. Filterkoeffizienten werden nicht mittels einer zuvor implementierten VHDL-Funktion ermittelt, sondern anhand der in Matlab definierten Funktionen fspecial() und freqz2() berechnet. Mit fspecial() werden die Koeffizienten zunächst im Zeitbereich ermittelt. Die Funktion bekommt in ihrem Argument ein Filter-Type 'gaussian', einen Integer-Wert hsize, der der Größe der Eingangsdaten (2D-DFT) entspricht, also 15 und als drittes Argument Sigma, die Standardabweichung. Als zurückgegebener Wert wird die Impulsantwort h im Zeitbereich ermittelt. Da die Filterung irrelevante Informationen oder Störgrößen im Frequenzbereich beseitigen soll, werden die mittels fspecial() ermittelte Impulsantwort im Frequenzdomain umgerechnet. Die Matlab-Funktion freqz2() ermöglicht den Berechnungsübergang der Koeffizienten von Zeit- in den Frequenzbereich. Es werden neben den Koeffizienten zwei Frequenzvektoren f1 und f2 normiert auf -1 bis +1 errechnet, die bei der Berechnung im Modul nicht berücksichtigt werden. Die Filterkoeffizienten werden mit den Eingangsdaten-Arrays des Sensors (8×8) und den Twiddle-Faktoren zum selben Zeitpunkt in das BRAM geladen. Abbildung zeigt 6.1 eine graphische Darstellung der ermittelten Koeffizienten.

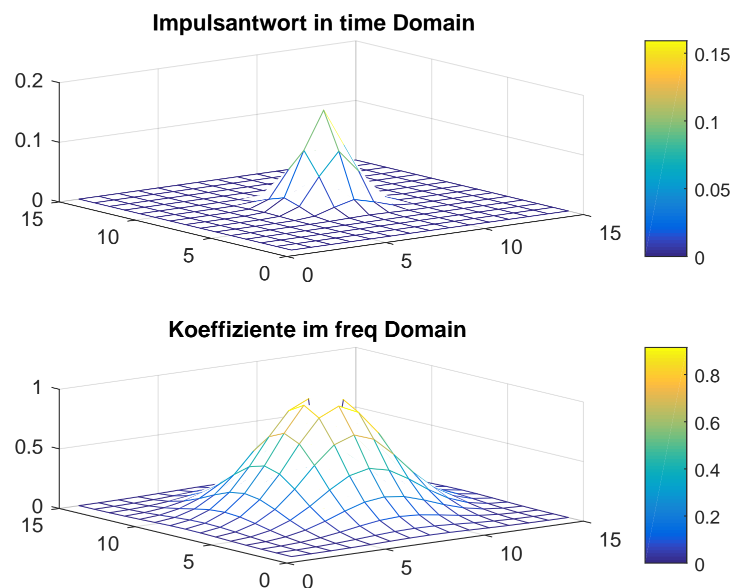


Abbildung 6.1: Ermittlung der Filterkoeffizienten über Matlab-freqz2()

Für die 225 ermittelten Koeffizienten, werden lediglich 120×32 Bit Speicherzellen

nötig. Die 15×12 Bit Werte aus einer Zeile der Koeffizienten-Matrix werden in `BRAM` sukzessiv paarweise gespeichert. Aus der folgenden Abbildung 6.2 wird die Paarbildung der Koeffizienten aus zwei Zeilen der Filter-Matrix graphisch dargestellt.

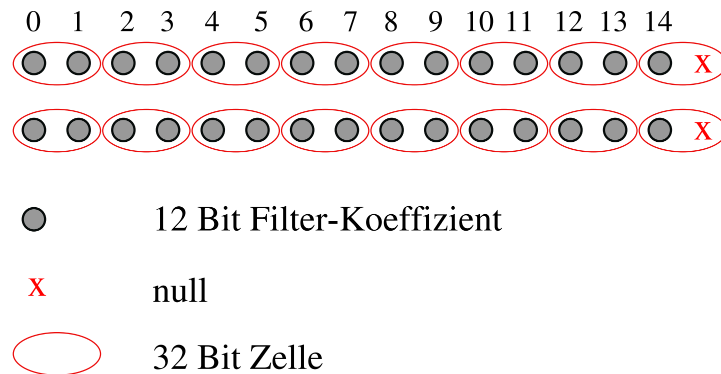


Abbildung 6.2: Die sequentielle Paarbildung der Filterkoeffizienten in 32 Bit Speicherzelle

6.0.1 Berechnungskonzept des Filters

Das hier zu implementierende 2D-Digitalfilter, soll die Multiplikation aus den im Frequenzbereich gebildeten Ergebnissen der 2D-DFT mit den der gespeicherten Filterkoeffizienten ausführen. Die Berechnungsvorgehensweise bei der Matrixmultiplikation ist zellenweise gestaltet. Die entsprechenden Matrixzellen werden miteinander multipliziert und das erzielte Ergebnis in das `BRAM` geschrieben. Beispielsweise würde die Multiplikation aus zwei quadratischen Matrizen der Dimension (3×3) solche Resultate liefern

$$\begin{bmatrix} \Phi_{11} & \Phi_{12} & \Phi_{13} \\ \Phi_{21} & \Phi_{22} & \Phi_{23} \\ \Phi_{31} & \Phi_{32} & \Phi_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \tag{6.1}$$

wobei

$$\Phi_{ij} = a_{ij} \cdot b_{ij} \tag{6.2}$$

ist. In dem jetzigen Modul soll die hier oben erwähnte Operation auf Matrizen der Dimension (15×15) erweitert werden.

6.0.2 Implementierung in VHDL

Wie in dem oben beschriebenen Modul gezeigt, ist die Implementierung der 2D-Digitalfilter in einer Darstellung mit drei Prozessen gebaut. Die wesentlichen Berechnungsschritte findet im `Process DataProcessing` statt. Da werden zuerst in einer sequentiellen Befehlsabarbeitung in dem Zählzustand `QCTNO` alle deklarierte Variable und Modul-Ausgang-Schnittstellen initialisiert. Anschließend werden innerhalb der nächsten fallenden Flanke

die gewählte Leseadresse und Readleitung angelegt bzw. aktiviert. Die ersten `32` Bit aus dem `BRAM` können in `real-` und `imgWert` separat und vorläufig gespeichert. Während derselben Taktflanke wird nach der Zerlegung und Speicheraktion der `32` Bit, die Adresse für das Auslesen der Filterkoeffizienten angelegt. Sie liefert zwei `12` Bit Realwerte, die unterschiedliche Speicherzelle ansprechen. Sie werden beide in zwei internen Variablen gespeichert Die Berechnung des ersten Ergebnis einer Speicherzelle kann durchgeführt werden. Bis zu diesem Zeitpunkt sind insgesamt `4 Takte` nötig. Da die Filterkoeffizienten reine reale Werte sind, erfolgt die Multiplikation als separater Produkt aus Filterkoeffizient mit Real- bzw. Imaginärteil der zuvor gespeicherten Daten. In einer sequentiellen Abarbeitung werden folgende Schritte ausgeführt:

- Die Readleitung deaktivieren und anschließend die nächste zu lesenden Datenadresse durch eine einfache Inkrementierung um eine Speicherzelle berechnen
- Die Schreibadresse an den Adressbus angelegen
- Die Aktivierung der Writeleitung
- Die Schreibaktion, das Übertragen der berechneten Ergebnisse in `BRAM` mittels `writeDATA()`

Die nächste Berechnungsphase erfordert keinen Zugriff im Speicherbereich der Filterkoeffizienten. Nur die Daten werden wie üblich ausgelesen und mit dem zweiten zuvor gespeicherten Koeffizient berechnet. Diese Routine soll solange durchlaufen werden, bis die letzte der Daten angesprochen und errechnet wird. Es folgt anschließend innerhalb eine `if-Abfrage` die Aktivierung des Ausgangs-Ready-Signal des Moduls und die Beendigung der Berechnung. Die Relevante Informationen für des Sensor-Array können aus einem bestimmten Speicherbereich selektiert und ausgelesen werden. Störgrößen werden zu Null umgewandelt.

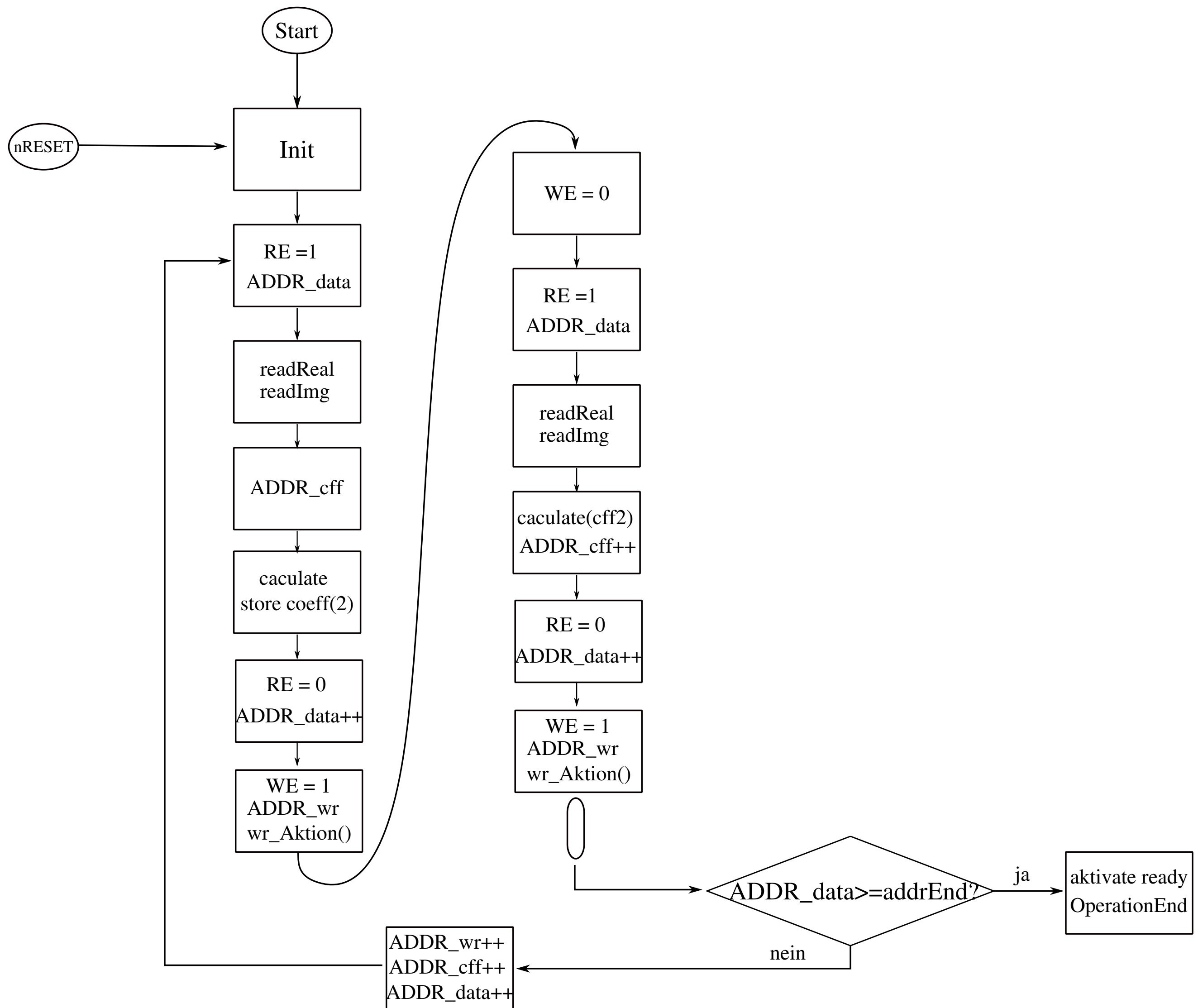


Abbildung 6.3: Programmablaufplan des 2D-Filters

7 Evaluation, Darstellung der Ergebnisse

In dem Kapitel werden die einzelnen erzielten Entwicklungsergebnisse auf dem `FPGA` graphisch dargestellt. anschließend folgen die Ergebnisse der Synthetisierung auf `FPGA`-Ebene und der Toolchain Cadence. Daraus kann man die Anzahl an Zellen feststellen , die die einzelnen Module im System generiert haben, d.h. die gesamte Zellenanzahl, die erzeugten logischen Gatter und die erwartete Chipfläche.

7.1 Evaluation der VHDL Interpolation-Modul

Abbildung 7.1 ist die graphische Darstellung der berechneten Interpolation auf VHDL FPGA-Ebene, verglichen mit der der Octave / Matlab-`interp2()`. Die optischen Unterschiede macht hier die für die Darstellung benutzte Octave-`imagesc()`. Eine 'bessere' Darstellung aus der optischen Sicht, schafft aus demselben Tool die Funktion `pcolor()`, das einzige Problem dabei ist, die unvollständige Repräsentation des gesamten Daten-Arrays. Die letzte Spalte der Daten wird bei der Darstellung mit `pcolor()` nicht betrachtet. Ansonsten kommt aus dem Vergleich der Ergebnisse eine Übereinstimmung der Daten sowohl der Realwerte als auch der Imaginärteile.

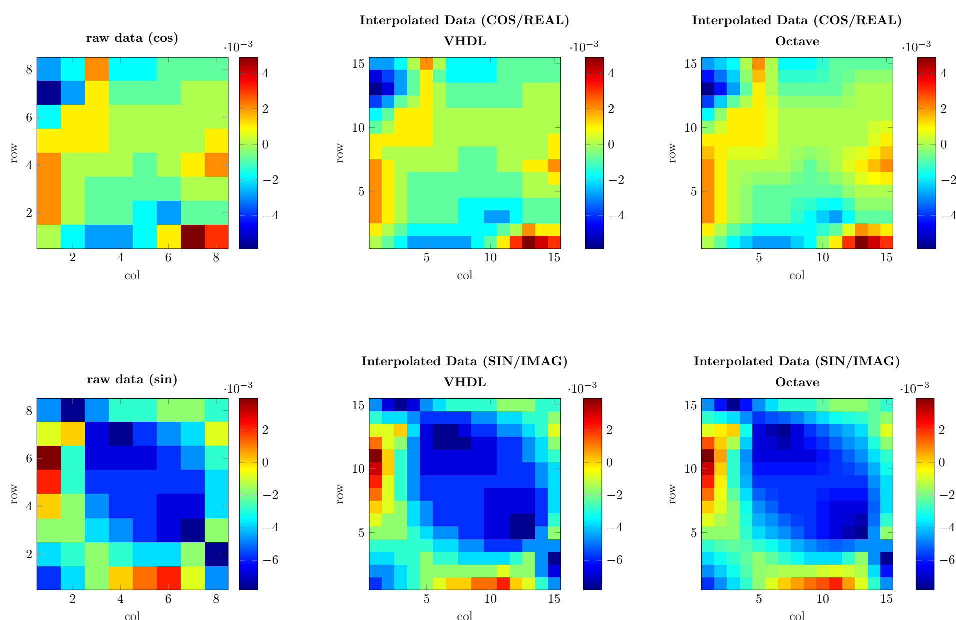


Abbildung 7.1: Ergebnisse der VHDL-Interpolation im Vergleich mit der Matlab Implementierung

7.2 Evaluation der VHDL 2D-DFT-Modul

Bei den Ergebnissen der `2D-DFT`; werden zunächst die Zwischen Ergebnisse, die `1D-DFT` hier gegenüber die vom Entwicklungstool Octave / Matlab in Real- und in Imaginärteil dargestellt. Folgende Abbildungen 7.2 und 7.3 für die `1D-DFT` bzw. `2D-DFT` ermöglichen eine bessere Analyse und Vergleich der Resultate.

7.2.1 Evaluation der VHDL 1D-MOD für 1D-DFT

Bis auf einige wenige Pixelwerte, die aus Grund der numerischen Berechnungen andere Repräsentation in Daten-Array aufweisen, wird der Vergleich der Ergebnisse aus einer globalen Sicht gut ausgewertet. Sowohl in `VHDL` als auch im `Octave` liegen die hier ausgerechneten Pixelwerte in demselben Wertebereich.

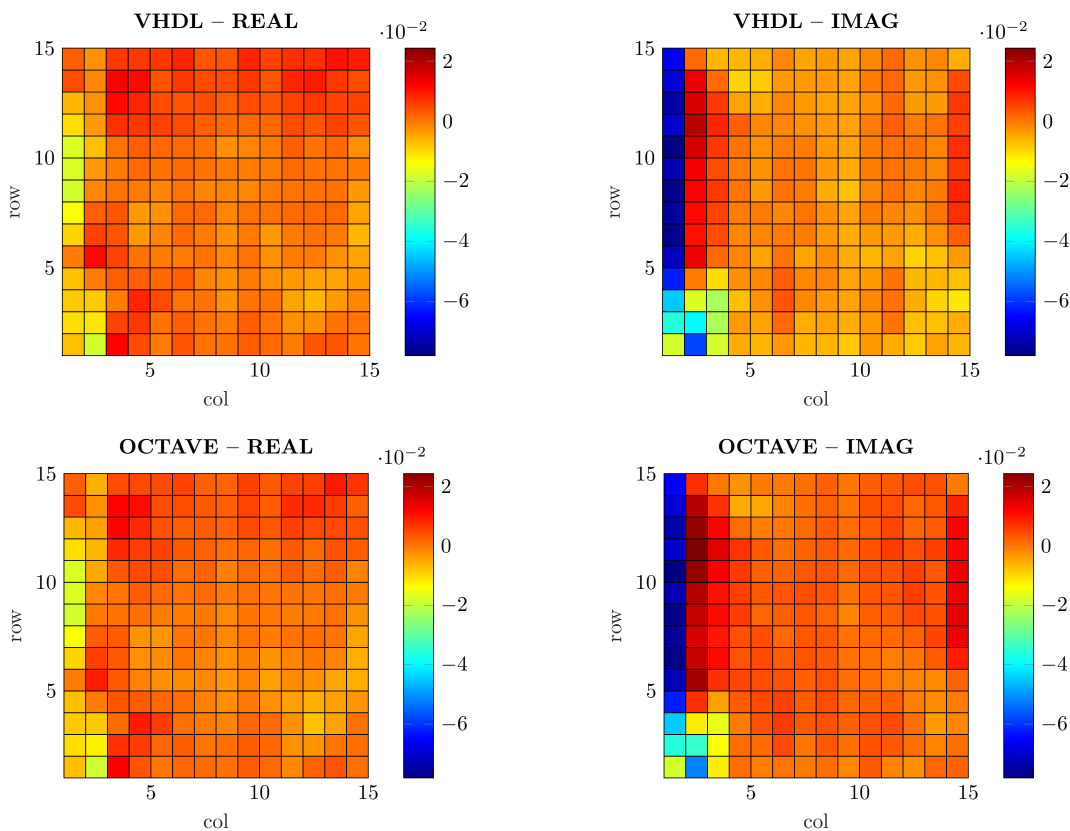


Abbildung 7.2: Ergebnisse der VHDL-Interpolation im Vergleich mit Octave- / Matlab-Implementierung

7.2.2 Evaluation der VHDL 1D-MOD für 2D-DFT

Identisch erwartet man bei der `2D-DFT` die gleichen Erscheinungen und Abweichungen der Pixelwerte wie in dem oberen Abschnitt. Die hier erzielten Ergebnisse hängen von

den Zwischenresultaten ab. Eine Numerische Probleme auf der `1D`-Ebene wird sich bei der Berechnung der `2D-DFT` übertragen und konnte diese Probleme erweitern. Eine geeignete Auswahl des Zahlenformats für die Darstellung der aktiven Werten könnte auf der Hardware-Ebene bessere Ergebnisse erzielen. Ein Implementierungsversuch mit dem Zahlenformat `Q11`, ein 12 Bit Darstellungsformat ohne Vorkommastelle und 11 Bit Nachkommastellen plus Vorzeichenbit könnten bessere Daten liefern.

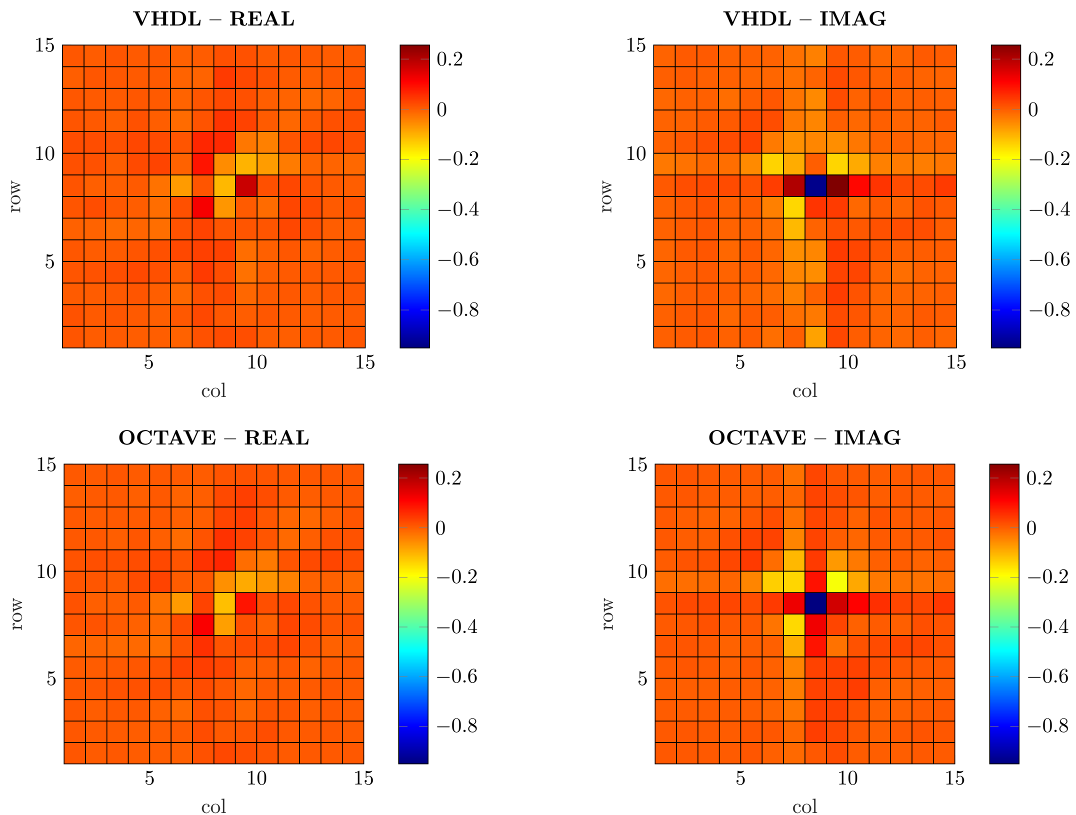


Abbildung 7.3: Ergebnisse der VHDL-Interpolation im Vergleich mit Octave- / Matlab-Implementierung

7.3 Evaluation der VHDL 2D-Filter

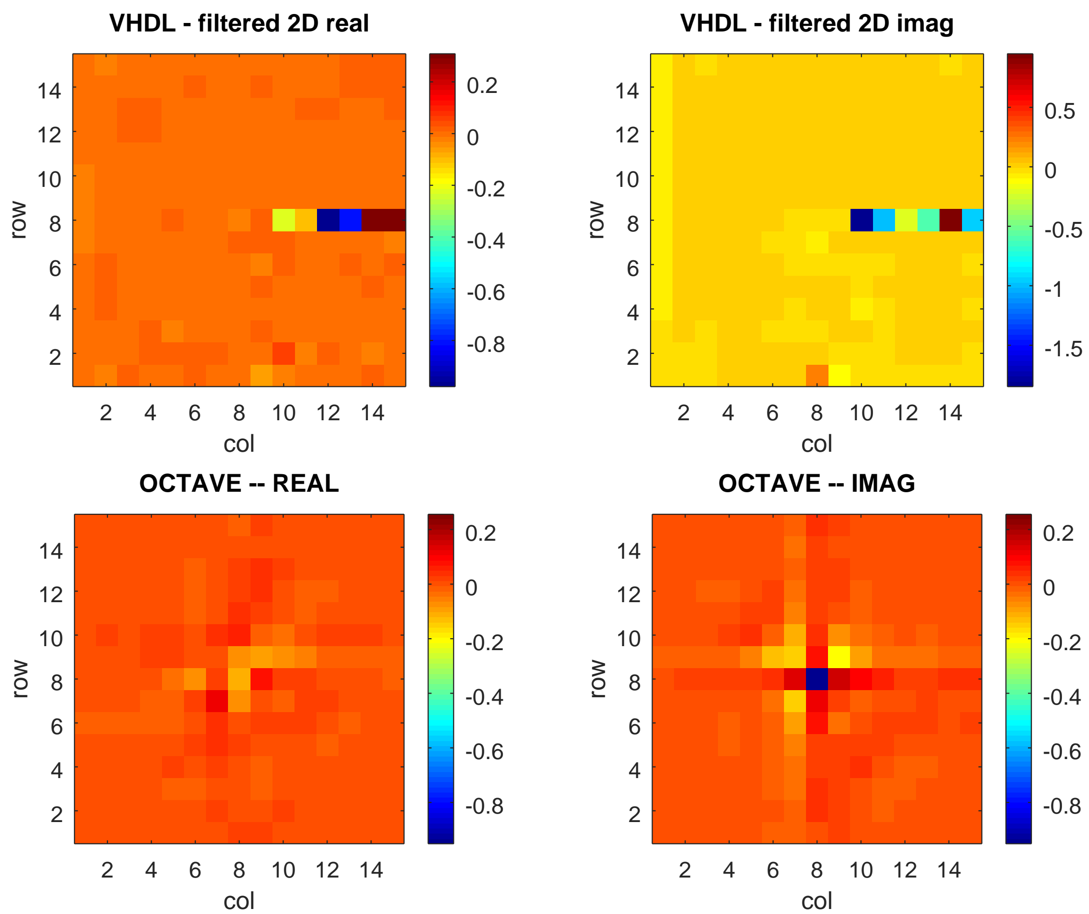


Abbildung 7.4: graphische Darstellung der Ergebnisse des Tiefpassfilers in VHDL

7.4 Synthese auf FPGA-Ebene, schematic

Report Cell Usage:

	Cell	Count
1	BUFG	5
2	CARRY4	388
3	DSP48E1	10
4	DSP48E1_1	2
5	LUT1	261
6	LUT2	1025
7	LUT3	365
8	LUT4	357
9	LUT5	365
10	LUT6	1218
11	MUXF7	21
12	RAMB36E1	1
13	FDCE	144
14	FDRE	960
15	FDSE	15
16	LDC	49
17	IBUF	19
18	OBUF	8
19	OBUFT	4

Report Instance Areas:

	Instance	Module	Cells
1	top		5217
2	C1	MEMORY_CONTROL	141
3	C2	BRAM	266
4	C3	MODULE_CONTROL	200
5	C4	INTERPOL2D	780
6	COLUMNINT	INT2D_COLUMN	369
7	ROWINT	INT2D_ROW	411
8	C5	ATAN2	641
9	C6	DD_FILTER	347
10	C7	zweiD_DFT	2802
11	DD_DFT	DD_MOD	1385
12	ADD2D	addUnit2D	309
13	OPERATE2D	operatUnit2D	1076
14	D_DFT	D_MOD	1417
15	ADD	addUnit	310
16	OPERATE	operatUnit	1107

Abbildung 7.5: Erzeugte Standardzellen auf Xilinx FPGA-Ebene

7.5 Synthese auf FPGA-Ebene, Evaluation der Standardzellen

```
Report Cell Usage:
+------+---------+------+
|      |Cell     |Count |
+------+---------+------+
|1     |BUFG     |     5|
|2     |CARRY4   |   388|
|3     |DSP48E1  |    10|
|4     |DSP48E1_1 |     2|
|5     |LUT1     |   261|
|6     |LUT2     |  1025|
|7     |LUT3     |   365|
|8     |LUT4     |   357|
|9     |LUT5     |   365|
|10    |LUT6     |  1218|
|11    |MUXF7    |    21|
|12    |RAMB36E1 |     1|
|13    |FDCE     |   144|
|14    |FDRE     |   960|
|15    |FDSE     |    15|
|16    |LDC      |    49|
|17    |IBUF     |    19|
|18    |OBUF     |     8|
|19    |OBUFT    |     4|
+------+---------+------+

Report Instance Areas:
+------+--------------+--------------+------+
|      |Instance      |Module        |Cells |
+------+--------------+--------------+------+
|1     |top           |              |  5217|
|2     |  C1          |MEMORY_CONTROL |   141|
|3     |  C2          |BRAM          |   266|
|4     |  C3          |MODULE_CONTROL |   200|
|5     |  C4          |INTERPOL2D    |   780|
|6     |    COLUMNINT |INT2D_COLUMN  |   369|
|7     |    ROWINT    |INT2D_ROW     |   411|
|8     |  C5          |ATAN2         |   641|
|9     |  C6          |DD_FILTER     |   347|
|10    |  C7          |zweiD_DFT     |  2802|
|11    |    DD_DFT    |DD_MOD        |  1385|
|12    |      ADD2D   |addUnit2D     |   309|
|13    |      OPERATE2D |operatUnit2D |  1076|
|14    |    D_DFT     |D_MOD         |  1417|
|15    |      ADD     |addUnit       |   310|
|16    |      OPERATE |operatUnit    |  1107|
+------+--------------+--------------+------+
```

Abbildung 7.6: Erzeugte Standardzellen auf Xilinx FPGA-Ebene

7.6 Synthese auf Cadence

```
73 OAI220            7      637.000    c35_CORELIB_TYP
74 OAI221            1       91.000    c35_CORELIB_TYP
75 OAI222          523    47593.000    c35_CORELIB_TYP
76 OAI310            6      546.000    c35_CORELIB_TYP
77 OAI311           17     1547.000    c35_CORELIB_TYP
78 XNR20            32     3494.400    c35_CORELIB_TYP
79 XNR21           624    68140.800    c35_CORELIB_TYP
80 XNR31            11     2202.200    c35_CORELIB_TYP
81 XOR20            12     1528.800    c35_CORELIB_TYP
82 XOR21             3      382.200    c35_CORELIB_TYP
83 XOR31             9     1801.800    c35_CORELIB_TYP
84 --------------------------------------------------
85 total          5513   701883.000
86
87
88
89      Type       Instances     Area     Area %
90 ----------------------------------------------
91 sequential           582 158576.600   22.6
92 inverter             583  21294.000    3.0
93 tristate              45   6552.000    0.9
94 logic               4303 515460.400   73.4
95 physical_cells         0      0.000    0.0
96 ----------------------------------------------
97 total               5513 701883.000  100.0
```

Abbildung 7.7: Erzeugte Standardzellen auf Cadence Chip-Ebene

8 Schlussfolgerungen

In diesem Kapitel wird zunächst im Abschnitt 8.1 eine ausführliche Zusammenfassung der gesamten Entwicklung eingegangen. Vor- und Nachteile des Entwicklungskonzepts bis zur Realisierung des Systems auf `FPGA` werden hier schrittweise erläutert anschließend folgen im Abschnitt 8.2 bessere Vorschläge, die hier aufgrund des begrenzten verfügbaren Zeitraums der Abschlussarbeit nicht umgesetzt und erprobt werden könnten. Beispielsweise Auswahl des Zahlenformats bzw. Bitbreite der aktiven Daten oder auf Leitungsebene, die Paarbildung (*WE, RE*) könnte auf eine einzige Leitung zusammengefasst werden.

8.1 Zusammenfassung

Die Arbeit leistet einen Teilbeitrag im Forschungsprojekt ISAR, das Signalverarbeitungsverfahren für magnetische Sensor-Arrays untersucht. Der Beitrag sollte exemplarisch prüfen, welcher Aufwand bei einer späteren Chipimplementierung für bestimmte Schritte der Signalverarbeitung anfällt. In dieser Arbeit wurden zunächst die Verfahren der Interpolation, die zweidimensionale Fouriertransformation und der Filterung über eine Matrixdarstellung theoretisch analysiert. Es wurde die digitale Zahlendarstellung und die Berechnung mit begrenzter Bitbreite betrachtet. Dann wurde eine Umgebung zur Erprobung auf einem FPGA in Betrieb genommen, diese basierte auf Vorarbeiten und wurde um drei umfangreiche Module erweitert. Diese Module umfassen:

- Die Interpolation einer 8x8 Matrix von Sensorwerten zu einer Matrix von 15x15 Zwischenwerten, die ebenfalls komplex sind.
- Die zweidimensionale Fouriertransformation für die Matrix von 15x15 Zwischenwerten. Dabei wurden zwei Schritte realisiert, welche jeweils eine eindimensionale Fouriertransformation durchführen. Die notwendigen Twiddle-Faktoren wurden extern errechnet und dem Modul effizient gespeichert zur Verfügung gestellt. Ebenso wurden zur Überprüfung auch die Werte der Stufen einzeln auslesbar bereitgestellt.
- Die Filterung der Werte mit einem beliebigen Tieffilter, das extern errechnet werden kann. Auch hier können die Werte zur Überprüfung ausgelesen werden.

Diese Module wurden in VHDL als Modelle beschrieben, simuliert und synthetisiert. Sie wurden auf einer Plattform, die aus einem Mikrocontroller (Texas Instrument TM4C1294) und einem FPGA-Board (Xilinx-Zedboard) besteht, in Betrieb genommen und praktisch erprobt. Zur Datenlieferung, zum Speicher schreiben und auslesen und zur

graphischen Darstellung dienten Programme im freien Mathematik-Tool Octave. Die Erprobungsergebnisse wurden bei Berechnung mit diesem Tool verglichen. Es gab bis auf geringe numerische Abweichung keine Differenzen zwischen den Octave-Berechnung und den Rechenwerten der eigenen Module auf dem FPGA. Weiterhin waren die graphischen Darstellung der Werte plausibel nachvollziehbar. Insgesamt wird die Umsetzung der drei Module als erfolgreich betrachtet.

8.2 Ausblick

Für die Zukunft werden eine Reihe von Anregungen für die Weiterentwicklung gegeben. Diese Vorschläge sollen in folgende Bereiche geordnet werden:

- Verbesserung der Speicherverwaltung
- Beschleunigung des Rechenverfahren
- Analyse der numerischen Effekte und Optimierung der Bitbreiten

Verbesserung der Speicherverwaltung

Die vorliegende Implementation speichert die Sensordaten (8x8×32Bit) als komplexe Werte und die Ergebnisse der nachfolgenden Interpolation (`15x15x32 Bit`) als komplexe Werte in separate Speicherbereiche. In den Ergebnissen der Interpolation sind die Sensordaten unverändert abgelegt, dazwischen die errechneten Werte. Diese separate Speicherung wurde gewählt, weil damit die Erprobung und die Entwicklung des Moduls Interpolation einfacher wurde. Nach der erfolgreichen Erprobung im Rahmen dieser Abschlussarbeit ist dieser Grund weggefallen. Nun ist die getrennte Speicherung der Sensordaten und die der Interpolationsergebnisse eher ungünstig, weil dieselben Werte im BRAM-Baustein vorkommen. Die Sensordaten (`8x8×32 Bit`) befinden sich doppelt gespeichert in zwei unterschiedlichen Speicherbereichen. Die Ressource BRAM wird auf dem Chip erhebliche Fläche erfordern, sie sollte sparsam verwendet werden.

Eine bessere Alternative wäre, dass bereits bei der Übertragung der Sensorrohdaten in den `BRAM` ein Speicherlücke von 32 Bit zwischen zwei sukzessiven Sensordaten frei gelassen wird. Danach wären für eine Zeile sieben Speicherlücken je 32 Bit für eine Datenreihe gespeichert. Immer wenn eine komplette Zeile der Eingangsmatrix im Speicher geschrieben ist, wird dann eine freie Zeile nicht belegt. Diese Speicherlücke der Größe (15×32Bit) liegt zwischen zwei aufeinander liegenden Zeilen.

In der Zukunft könnte ein leicht überarbeites Interpolations-Modul die Sensordaten nicht mehr kopieren. Die Ergebnisse der Interpolationsrechnung würden dann in die freigelassenen Speicherlücken eingetragen.

Diese Verbesserung hätte zwei Vorteile und einen Nachteil. Der erste Vorteil ist die Einsparung und Freisetzung von BRAM-Speicherplätzen: Statt 289 × 32Bit werden nur

225×32 Bit benötigt. Der zweite Vorteil ist die Einsparung von 64 x 32 Bit Schreiboperationen, welche nur Sensorwerte kopieren. Das führt zu einer Reduzierung der Anzahl der Takte (mindestens 2×64 Takte) und zu einer Erhöhung der Rechengeschwindigkeit. Der Nachteil sind die erschwerten Test-und Debugging-Möglichkeiten, die aber nur in der Entwicklungsphase ein Problem darstellen und nach der Fertigstellung kaum erheblich sind.

Die Speicherverwaltung kann auch in anderen Aspekten noch optimiert werden. Die wichtige Entscheidung der 2D-DFT-Implementierung ist, wie man mit den 225 Twiddle-Faktoren effizient umgeht.

In der Entwicklungsphase war es aber zunächst sinnvoll, die Twiddle-Faktoren extern zu berechnen und in den `BRAM` zu speichern. Es wurde jedoch die Symmetrie und Wiederholung gleicher Faktoren bereits genutzt. Es müssen nicht unbedingt die ganzen Matrizen mit (2×225) Werten in den BRAM geschrieben werden, da sich viele Koeffizienten oder Zeilen der Twiddle-Matrix wiederholen.

In der vorliegenden Arbeit wurde eine Vereinfachung bereits angewandt, in dem nur der erste Quadrant der Matrix gespeichert wird. Das hat zwei Vorteile, Vermeidung von mehrfacher Speicherung im BRAM und eine erhörte Berechnungsgeschwindigkeit durch weniger Leseoperationen des Algorithmus. Der Nachteil liegt in der erheblichen Komplikation bei Adressierung im Speicherbereich der Twiddle-Faktoren.

Eine Alternative, die vor der Implementierung des Moduls 2D-DFT überlegt und abgeschätzt wurde, ist die Speicherung der ersten zwei hintereinander liegenden Quadranten der Twiddlefaktor-Matrizen.

Dabei werden die Adressierungsvorgänge der einzelnen Faktoren wesentlich weniger kompliziert ausgeführt, weil die ausgewählte Adresse lediglich um eine Speicherzelle für den Lesevorgang inkrementiert werden muss.

Eine letztlich mögliche 'harte' Codierung der einzelnen Faktoren auf Chipebene würde keinen Speicher im BRAM erfordern. Alle Lesevorgänge für Twiddlefaktoren entfallen und damit die benötigte Zeit. Es kommen dann Festwert-Multiplikations-Einheiten für jeden Twiddelfaktorwert zum Einsatz, leider ist diese Methode jedoch sehr unflexibel. Beispielsweise würde jede Matrixgröße vollständig andere Faktoren und Multiplizierer erfordern. Ebenso sind die Festwert-Multiplizierer fest an die Bitbreiten der Rechnung gebunden. Weiterhin würde die Implementierung VHDL wahrscheinlich komplizierter werden. Dennoch wird in dieser Berechnungsmethode mit Festwert-Multiplzierern sehr viel Potential an Flächen- und Zeiteinsparung gesehen.

Beschleunigung des Rechenverfahren

In der vorliegenden Arbeit wird nur in beschränkten Umfang parallel gearbeitet, beispielweise werden Real- und Imaginärergebnisse gemeinsam in gleichen Takt verarbeitet. Jedes Ergebniss wird z.B. in einer DFT-Stufe nacheinander allgemeinen in 46 Takten errechnet. Viele Takte sind für Leseoperationen am BRAM erforderlich.

Durch Parallelarbeit kann hier viel eingespart werden. Zunächst bieten sich Pipelineverfahren an, die auf mehren Ebenen implementiert werden können:

- Ebene der Einzelwertberechnung
- Ebene der beiden Stufen der DFT
- Ebene der obersten Stufen der Interpolation, DFT und Filterung

Die Voraussetzung dafür ist die Verteilung der gemeinsamen Speicher-Ressource BRAM. Beispielsweise wären getrennte Übergabespeicher zwischen Interpolation und DFT und Filterung zu realisieren, ebenso wäre das zwischen den Stufen der DFT möglich. Es sollte jedoch darauf geachtet werden, das alle Stufen etwa die gleiche Anzahl an Takten benötigen.

Eine Pipeline wäre bereits durch weniger Leseoperation mit Festwert-Multiplizierern (siehe zuvor) leichter auf der Ebene der Einzelwertberechnung zu gestalten.

Allerdings ist der Entwicklungsprozess bei Parallelarbeit komplizierter. Es ist zwar von Zeiteinsparung auszugehen, jedoch nicht von Flächenreduzierung.

Analyse der numerischen Effekte und Optimierung der Bitbreiten

Die Festlegung der Bitbreiten erfolgte in dieser Arbeit durch Abschätzungen und Vorgaben. Grundlagen war die erwartete Bitbreite des ADC (10-12Bit) und die gewünschte Genauigkeit von einen Winkel-Grad (entsprechend etwa 0,25 Prozent). Es standen nur simulierte Werte oder Werte aus Demonstrations-Arrays mit 50-25 Vergrößerung und anderen Sensortypen zu Verfügung. Es liegen noch keine konkreten Sensorwerte und deren Toleranzen vor. Ebenso ist der endgültige Algorithmus noch in der Entwicklung. Daraus ergibt sich auch die Frage der verbleibenden numerischen Fehler. Zur Zeit kann nicht abgeschätzt werden, ob mehr oder weniger Bitbreite erforderlich ist oder ob dieses nur für bestimmte Rechnungen gilt.

Es ist zu erwarten, dass es zu einer gründlichen Überprüfung der Bitgenauigkeiten kommen muss. Daraus können sich weitere Optimierungen ergeben.

Literatur

[1] Tilman Butz. *Fouriertransformation für Fußgänger*. Springer, 2009.

[2] Klaus Fricke. *Digitaltechnik: Lehr-und Übungsbuch für Elektrotechniker und Informatiker*. Springer-Verlag, 2010.

[3] Winfried Gehrke u. a. *Digitaltechnik: Grundlagen, VHDL, FPGAs, Mikrocontroller*. Springer-Verlag, 2016.

[4] Ralf Gessler. *Entwicklung Eingebetteter Systeme.*

[5] Rainer Kelch. *Rechnergrundlagen. Von der Binärlogik zum Schaltwerk*. Hanser Verlag, 2003.

[6] Hans Peter Kölzer. *Bildverarbeitung: Vorlesung Elektrotechniker und Informatiker*. HAW-Hamburg, 2017.

[7] Dok Won Lee, William David French und Keith Ryan Green. *Integrated circuit with hall effect and anisotropic magnetoresistive (AMR) sensors*. US Patent 9,893,119. Feb. 2018.

[8] Prof.Dr.C.Clemen. *Digitale Filter*. Bd. 11. FH Augsburg, 1999.

Anhang

A Quellcode

Quellcode A.1: DSP_PKG.vhd: Definitionen von Konstanten.

```
--------------------------------------------------------------------------------
-- Company: Diese Entity ist ein Teil-Komponent des 2D-DFT Modul
-- ist sowohl in 1D-DFT als auch invertiert i 2D-DFT eingesetzt
-- berechnet die 210 Speicherzelle des BRAMs.
-- Engineer:
--
-- Create Date: 10/04/2018 04:38:59 PM
-- Design Name: Ada Koundoul Mastermikroelektronik
-- Module Name: Package - Behavioral
-- Project Name: Masterarbeit ISAR
-- Target Devices: Zedboard Xilinx
-- Tool Versions:
-- Description: Package Komponent
--
-- Dependencies:
--
-- Revision: Hamburg 24.10.2018
-- Revision 0.01 - File Created
-- Additional Comments:
--
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.Numeric_std.all;
--==============================================================
--%%%%%%%%%%%%%%%%%%%%%%%%PACKAGE_DSP%%%%%%%%%%%%%%%%%%%%%%%%%
--==============================================================
package DSP_PKG is
-- %%%%% KONSTANTENDEKLARATION %%%%%
constant DRAM_WIDTH       : natural := 32;
constant MOD_WIDTH        : natural := 3;
constant QWIDTH           : natural := 6;
constant ADDR_WIDTH       : natural := 10;
constant DATA_WIDTH       : natural := 12;
constant DWIDTH16         : natural := 16;
constant ST_WIDTH         : natural := 3;
constant PDC_WIDTH        : natural := 24;
--
--%%%%%%%%%%%%% RE, WE VALUE %%%%%%%%%%%%%%
constant HIGHLEVEL        : STD_LOGIC := '1';
constant LOWLEVEL         : STD_LOGIC := '0';
--%%%%%%%%%%%%%% DATA_LIMIT %%%%%%%%%%%%%%
constant R_MSB            : natural := 31;
constant R_LSB            : natural := 20;
--
constant I_MSB            : natural := 15;
constant I_LSB            : natural := 4;
-- %%%%%%%%%%%%%%SUBTYPE_DEFINITION%%%%%%%%%%%%%%
subtype SLV               is STD_LOGIC_VECTOR(DRAM_WIDTH-1 downto 0);      -- DATA_TYPE FROM RAM 32
       BITS
subtype ADDR_TYPE         is STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto 0);
subtype RVALUE_TYPE       is STD_LOGIC_VECTOR(DWIDTH16-1 downto 0);
--%%%%%%%%%%%%%%%%%% ADDR_VALUE %%%%%%%%%%%%%%%%%%
constant NULL_4SLV        : std_logic_vector(3 downto 0) := "0000";
constant ST_RE_ADDR       : ADDR_TYPE := "0000000000";
constant ST_WE_ADDR       : ADDR_TYPE := "0001000000";     -- 64  store interpolationsData ab hier.
       || start addr0
constant ST_ADD_READ1     : ADDR_TYPE := "0001011110";     -- 94  start_addr1
constant ST_ADDR_WRITE    : ADDR_TYPE := "0001001111";     -- 79  -- write ab hier
constant END_RHO_ADDR     : ADDR_TYPE := "0000111111";
--
subtype DATA_TYPE         is signed(DATA_WIDTH-1 downto 0);
subtype PDC_TYPE          is signed(PDC_WIDTH-1 downto 0);
-- %%%%%%%%%%%%%%%%%% CTRL_VALUE %%%%%%%%%%%%%%%%%%
subtype  MODUL            is STD_LOGIC_VECTOR(MOD_WIDTH-1 downto 0);
constant INTMOD           : MODUL     := "001";                  -- aktivate INTERPOLATIONS_MODUL
constant DFTMOD           : MODUL     := "010";                  -- aktivate DFT_MODUL
constant TPMOD            : MODUL     := "100";                  -- aktivate FILTER_MODUL
--%%%%%%%%%%%%%%%%%%%% QCOUNTER_VALUE %%%%%%%%%%%%%%%%%%%%
subtype QCOUNTER          is unsigned(QWIDTH-1 downto 0);          -- 6 BITS BREITER
constant QCTN0            : QCOUNTER := "000000";    -- 0
constant QCTN1            : QCOUNTER := "000001";    -- 1
```

```
    constant QCTN2          : QCOUNTER := "000010";   -- 2
    constant QCTN3          : QCOUNTER := "000011";   -- 3
    constant QCTN4          : QCOUNTER := "000100";   -- 4
    constant QCTN5          : QCOUNTER := "000101";   -- 5
75  constant QCTN6          : QCOUNTER := "000110";   -- 6
    constant QCTN7          : QCOUNTER := "000111";   -- 7
    constant QCTN8          : QCOUNTER := "001000";   -- 8
    constant QCTN9          : QCOUNTER := "001001";   -- 9
    constant QCTN10         : QCOUNTER := "001010";   -- 10
80  constant QCTN11         : QCOUNTER := "001011";   -- 11
    constant QCTN12         : QCOUNTER := "001100";   -- 12
    constant QCTN13         : QCOUNTER := "001101";   -- 13
    constant QCTN14         : QCOUNTER := "001110";   -- 14
    constant QCTN15         : QCOUNTER := "001111";   -- 15
85  constant QCTN16         : QCOUNTER := "010000";   -- 16
    constant QCTN17         : QCOUNTER := "010001";   -- 17
    constant QCTN18         : QCOUNTER := "010010";   -- 18
    constant QCTN19         : QCOUNTER := "010011";   -- 19
    constant QCTN20         : QCOUNTER := "010100";   -- 20
90  constant QCTN21         : QCOUNTER := "010101";   -- 21
    constant QCTN22         : QCOUNTER := "010110";   -- 22
    constant QCTN23         : QCOUNTER := "010111";   -- 23
    constant QCTN24         : QCOUNTER := "011000";   -- 24
    constant QCTN25         : QCOUNTER := "011001";   -- 25
95  constant QCTN26         : QCOUNTER := "011010";   -- 26
    constant QCTN27         : QCOUNTER := "011011";   -- 27
    constant QCTN28         : QCOUNTER := "011100";   -- 28
    constant QCTN29         : QCOUNTER := "011101";   -- 29
    constant QCTN30         : QCOUNTER := "011110";   -- 30
100 constant QCTN31         : QCOUNTER := "011111";   -- 31
    constant QCTN32         : QCOUNTER := "100000";   -- 32
    constant QCTN33         : QCOUNTER := "100001";   -- 33
    constant QCTN34         : QCOUNTER := "100010";   -- 34
    constant QCTN35         : QCOUNTER := "100011";   -- 35
105 constant QCTN36         : QCOUNTER := "100100";   -- 36
    constant QCTN37         : QCOUNTER := "100101";   -- 37
    constant QCTN38         : QCOUNTER := "100110";   -- 38
    constant QCTN39         : QCOUNTER := "100111";   -- 39
    constant QCTN40         : QCOUNTER := "101000";   -- 40
110 constant QCTN41         : QCOUNTER := "101001";   -- 41
    constant QCTN42         : QCOUNTER := "101010";   -- 42
    constant QCTN43         : QCOUNTER := "101011";   -- 43
    constant QCTN44         : QCOUNTER := "101100";   -- 44
    constant QCTN45         : QCOUNTER := "101101";   -- 45
115 constant QCTN46         : QCOUNTER := "101110";   -- 46
    constant QCTN47         : QCOUNTER := "101111";   -- 47
    constant QCTN48         : QCOUNTER := "110000";   -- 48
    constant QCTN49         : QCOUNTER := "110001";   -- 49
    constant QCTN50         : QCOUNTER := "110010";   -- 50
120 constant QCTN51         : QCOUNTER := "110011";   -- 51
    constant QCTN52         : QCOUNTER := "110100";   -- 52
    constant QCTN53         : QCOUNTER := "110101";   -- 53
    constant QCTN54         : QCOUNTER := "110110";   -- 54
    constant QCTN55         : QCOUNTER := "110111";   -- 55
125 constant QCTN56         : QCOUNTER := "111000";   -- 56
    constant QCTN57         : QCOUNTER := "111001";   -- 57
    constant QCTN58         : QCOUNTER := "111010";   -- 58
    constant QCTN59         : QCOUNTER := "111011";   -- 59
    constant QCTN60         : QCOUNTER := "111100";   -- 60
130 constant QCTN61         : QCOUNTER := "111101";   -- 61
    constant QCTN62         : QCOUNTER := "111110";   -- 62
    constant QCTN63         : QCOUNTER := "111111";   -- 62
    --
    end DSP_PKG;
135 --==========================================================
    --*************PACKAGE_BODY_DSP***********
    --==========================================================
    package body DSP_PKG is
    end DSP_PKG;
```

Quellcode A.2: DFT_FUNCTIONS.vhd: Funktionen für die 2D-DFT.

```
    ----------------------------------------------------------------------------------
    -- Company: Diese Entity ist ein Teil-Komponent des 2D-DFT Modul
    -- ist sowohl in 1D-DFT als auch invertiert i 2D-DFT eingesetzt
    -- berechnet die 210 Speicherzelle des BRAMs.
5   -- Engineer:
    --
    -- Create Date: 10/04/2018 04:38:59 PM
    -- Design Name: Ada Koundoul Mastermikroelektronik
    -- Module Name: Package - Behavioral
10  -- Project Name: Masterarbeit ISAR
    -- Target Devices: Zedboard Xilinx
    -- Tool Versions:
    -- Description: Package Komponent
```

```
   --
15 -- Dependencies:
   --
   -- Revision: Hamburg 24.10.2018
   -- Revision 0.01 - File Created
   -- Additional Comments:
20 --
   ----------------------------------------------------------------------------------
   library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
   use ieee.Numeric_std.all;
25 --use ieee.std_logic_unsigned.all;
   use work.DSP_PKG.all;
   use work.FUNCTIONS_PKG.all;

   package DFT_FUNCTIONS is

      --        SPEICHERBEREICH
30 constant data_msb                    : integer                    := 11;
   --DFT_ADDR_CONSTANTE
   constant start_addr_down             : ADDR_TYPE                  := "0100010010";
                                   -- 274  read interpolationsdata
   constant start_addr_up               : ADDR_TYPE                  := "0001001111";
                                   -- 79   read interpolationsdata (zweitzeilenanfang)
   constant st_addr_1d                  : ADDR_TYPE                  := "0100110000";
                                   -- 304  die zweite Zeile 1D_DFT_Results schreiben
35 constant addr_2d                     : ADDR_TYPE                  := "1000000010";
                                   -- 514  2d_start_addr
   constant TW_st_addr                  : ADDR_TYPE                  := "1011100011";
                                   -- 739  store from here the 225 Twiddle-Factors
   constant col0_addr                   : ADDR_TYPE                  := "1000010001";
                                   -- 529  Write_Start_addr Modul COL0DFT
   constant end_rowAddr                 : ADDR_TYPE                  := "0100101111";
                                   -- 303
   constant forward_addr                : ADDR_TYPE                  := "0100110001";
                                   -- 305  zweite Zelle der zweiten Zeile der 1D_DFT
40 constant backward_addr               : ADDR_TYPE                  := "0100111110";
                                   -- 318  letze Zelle der zweiten Zeile der 1D_DFT
   constant addr2write                  : ADDR_TYPE                  := "1000010010";
                                   -- 530  ab hier den Rest der 2D_DFT-Zellen ausf\"ullen
   constant row_addr_cpy                : ADDR_TYPE                  := "1000010000";
                                   -- 528
   constant Qdrtaddr                    :  ADDR_TYPE                 := "1000011111";
                                   -- 543                          --
   constant fCol                        : unsigned(3 downto 0)       := "0000";
                                        --      FIRST COLUMN DER TW-MATRIX
45 -- TW addr-ROW
   constant TWrow1                      : ADDR_TYPE                  := "1011101010";
                                   -- 746
   constant TWrow2                      : ADDR_TYPE                  := "1011110001";
                                   -- 753
   constant TWrow3                      : ADDR_TYPE                  := "1011111000";
                                   -- 760
   constant TWrow4                      : ADDR_TYPE                  := "1011111111";
                                   -- 767
50 constant TWrow5                      : ADDR_TYPE                  := "1100000110";
                                   -- 774
   constant TWrow6                      : ADDR_TYPE                  := "1100001101";
                                   -- 781
   --
   constant max              : data_type              := "011111111111";
   constant min                         : data_type                  := "100000000000
       ";
55 --
   constant nullvalue                   : DATA_TYPE                  := "000000000000
       ";                          -- complex_mult result with null
   -- DFT_FUNCTIONS-PROTOTYPE
   function RUN_COL_FORWARD(ADDR        : ADDR_TYPE)                 return ADDR_TYPE
       ;                           -- laeuft die Matrix Spaltenweise runter(steigend)
   function jumpaddrback(ADDR           : ADDR_TYPE)                 return ADDR_TYPE
       ;                           -- laeuft die Matrix Spaltenweise hoch  (fallend)
60 function GO2NEXT_UPCOL(ADDR          : ADDR_TYPE)                 return ADDR_TYPE
       ;
   function GO2NEXT_DOWNCOL(ADDR        : ADDR_TYPE)                 return ADDR_TYPE
       ;
   --function jump2nextaddr(ADDR          : ADDR_TYPE)                return
       ADDR_TYPE;
   function jump2_1row(ADDR             : ADDR_TYPE)                 return ADDR_TYPE
       ;
   function twoComplement(arg           : DATA_TYPE)                 return DATA_TYPE
       ;
65 function jump2CopyPosition (arg      : ADDR_TYPE; n : unsigned(2 downto 0))  return ADDR_TYPE
       ;                           -- berechnet die Addr, f\"ur die copy-paste Data
   function back2WritePosition(arg      : ADDR_TYPE; n : unsigned(2 downto 0))  return ADDR_TYPE
       ;
   function back2startPoint(arg         : ADDR_TYPE)                 return ADDR_TYPE
       ;
   function go2startPoint(arg           : ADDR_TYPE)                 return ADDR_TYPE
```

```
        ;
    function startWNwriteRow(arg          : ADDR_TYPE)                        return ADDR_TYPE
        ;
70  function return2LastCol(arg          : ADDR_TYPE)                        return ADDR_TYPE
        ;
    function settingTWiddle(n : unsigned(2 downto 0))                        return ADDR_TYPE
        ;
    --========================================
    function additionOf12Bit(value1       : DATA_TYPE; value2       : DATA_TYPE)  return DATA_TYPE
        ;
    function subtractionOf12Bit(value1    : DATA_TYPE; value2       : DATA_TYPE)  return DATA_TYPE
        ;
75  --============================
    function signExtension(arg          : DATA_TYPE)                         return signEXT;
    function calcPDC(DATA   :DATA_TYPE; TWFAKT : DATA_TYPE)                   return DATA_TYPE
        ;
    end package;
    --
80  package body DFT_FUNCTIONS is
    --========================================
    -- ROW OBEN NACH UNTEN(STEIGEND)
    --========================================
    function RUN_COL_FORWARD(ADDR          : ADDR_TYPE)                      return ADDR_TYPE
         is
85      variable RSLT        : ADDR_TYPE := (others => '0');
    begin
        RSLT     := std_logic_vector(unsigned(ADDR)+ 15);          --
        return RSLT;
    end function;
90  --========================================
    -- ROW UNTEN NACH OBEN(FALLEND)
    --========================================
    function jumpaddrback(ADDR          : ADDR_TYPE)                         return ADDR_TYPE
         is
        variable RSLT        : ADDR_TYPE := (others => '0');
95  begin
        RSLT     := std_logic_vector(unsigned(ADDR)- 15);
        return RSLT;
    end function;
    --========================================
100 -- go to neben stehender Saplte und run fallend (UP_addr)
    --========================================
    function GO2NEXT_UPCOL(ADDR         :ADDR_TYPE)                          return ADDR_TYPE
         is
        variable RSLT        : ADDR_TYPE := (others => '0');
    begin
105     RSLT     := std_logic_vector(unsigned(ADDR)- 89);
        return RSLT;
    end function;
    --========================================
    -- go to neben stehender Saplte und run fallend (down_addr)
110 --========================================
    function GO2NEXT_DOWNCOL(ADDR         :ADDR_TYPE)                        return ADDR_TYPE
         is
        variable RSLT        : ADDR_TYPE := (others => '0');
    begin
        RSLT     := std_logic_vector(unsigned(ADDR)+ 91);
115     return RSLT;
    end function;
    --========================================
    -- BERECHNE PDCT AUS SUMME UND TWIDDLE_FAKTOR
    -- Betrag berechnen
120 -- Product rechnen
    -- Signe anpassen
    --========================================
    function calcPDC(DATA            : DATA_TYPE; TWFAKT : DATA_TYPE)                  return
        DATA_TYPE is
            -- die Multiplikation nur gultig wenn arg < 1
125 variable opRslt           : PDC_TYPE        := (others => '0');
    variable output           : data_type        := (others => '0');
    begin
    opRslt    := PDC_TYPE (DATA*TWFAKT);
    output    := data_type(opRslt(pdc_width-1)&opRslt(pdc_width-4 downto 10));
130 return output;
    end function;
    --========================================
    -- jump2nextaddr
    --========================================
135 --function jump2nextaddr(ADDR              : ADDR_TYPE)                      return
        ADDR_TYPE is
    --     variable rslt            : ADDR_TYPE := (others => '0');
    --begin
    --     rslt     := std_logic_vector(unsigned(ADDR)+15);
    --     return rslt;
140 --end function;
    --========================================
    -- jump to first row
    --========================================
```

```
    function jump2_1row(ADDR                : ADDR_TYPE)                    return ADDR_TYPE
        is
145     variable rslt           : ADDR_TYPE := (others => '0');
    begin
        rslt                := std_logic_vector(unsigned(ADDR)-209);
        return rslt;
    end function;
150 --===================================
    -- build two Complement of 12 bit data
    --===================================
    function twoComplement(arg:             data_type)                      return data_type
        is
        variable rslt       : data_type := (others => '0');
155 begin
        rslt    := (not(arg) + 1);
        return rslt;
    end function;
    --===================================
160 -- Address for Copy-Paste
    --===================================
    function jump2CopyPosition (arg: ADDR_TYPE; n : unsigned(2 downto 0))       return ADDR_TYPE
        is
    variable rslt           : ADDR_TYPE := (others =>'0');
    begin
165     case to_integer(n) is
            when 0 =>
                rslt    := std_logic_vector(unsigned(arg)+195);
            when 1 =>
                rslt    := std_logic_vector(unsigned(arg)+165);
170         when 2 =>
                rslt    := std_logic_vector(unsigned(arg)+135);
            when 3 =>
                rslt    := std_logic_vector(unsigned(arg)+105);
            when 4 =>
175             rslt    := std_logic_vector(unsigned(arg)+75);
            when 5 =>
                rslt    := std_logic_vector(unsigned(arg)+45);
            when 6 =>
                rslt    := std_logic_vector(unsigned(arg)+15);
180         when others => null;
        end case;
        return rslt;
    end function;
    --===================================
185 -- Positioniert die wr_addr after copy-paste
    --===================================
    function back2WritePosition(arg :ADDR_TYPE; n : unsigned(2 downto 0))       return ADDR_TYPE
        is
    variable rslt           : ADDR_TYPE := (others =>'0');
    begin
190     case to_integer(n) is
            when 0 =>
                rslt    := std_logic_vector(unsigned(arg)-180);
            when 1 =>
                rslt    := std_logic_vector(unsigned(arg)-150);
195         when 2 =>
                rslt    := std_logic_vector(unsigned(arg)-120);
            when 3 =>
                rslt    := std_logic_vector(unsigned(arg)-90);
            when 4 =>
200             rslt    := std_logic_vector(unsigned(arg)-60);
            when 5 =>
                rslt    := std_logic_vector(unsigned(arg)-30);
            when others => null;
        end case;
205     return rslt;
    end function;
    --===================================
    -- forward_addr als argument
    --===================================
210 function settingTWiddle(n : unsigned(2 downto 0))  return ADDR_TYPE is
    variable rslt           : ADDR_TYPE := (others =>'0');
    begin
        case to_integer(n) is
            when 0 =>
215             rslt    := TW_st_addr;                                  -- 739
            when 1 =>
                rslt    := std_logic_vector(unsigned(TW_st_addr)+7);    --746
            when 2 =>
                rslt    := std_logic_vector(unsigned(TW_st_addr)+14);   -- 753
220         when 3 =>
                rslt    := std_logic_vector(unsigned(TW_st_addr)+21);   -- 760
            when 4 =>
                rslt    := std_logic_vector(unsigned(TW_st_addr)+28);   -- 767
            when 5 =>
225             rslt    := std_logic_vector(unsigned(TW_st_addr)+35);   -- 774
            when 6 =>
                rslt    := std_logic_vector(unsigned(TW_st_addr)+42);   -- 781
```

```
            when others => null;
        end case;
230     return rslt;
    end function;
    --==============================================
    -- forward_addr als argument
    --==============================================
235 function back2startPoint(arg :ADDR_TYPE)                         return ADDR_TYPE
        is
        variable rslt          : ADDR_TYPE := (others => '0');
    begin
        rslt              := std_logic_vector(unsigned(arg)-6);
        return rslt;
240 end function;
    --==============================================
    -- backward_addr als argument
    --==============================================
    function go2startPoint(arg :ADDR_TYPE)                           return ADDR_TYPE
        is
245     variable rslt          : ADDR_TYPE := (others => '0');
    begin
        rslt              := std_logic_vector(unsigned(arg)+6);
        return rslt;
    end function;
250 --==============================================
    -- a new write ROW
    --==============================================
    function startWNwriteRow(arg          : ADDR_TYPE)               return ADDR_TYPE
        is
        variable rslt          : ADDR_TYPE := (others => '0');
255 begin
        rslt              := std_logic_vector(unsigned(arg)+2);
        return rslt;
    end function;
    --==============================================
260 -- two 12Bits Addition
    --==============================================
    function additionOf12Bit(value1       : DATA_TYPE; value2       : DATA_TYPE)  return DATA_TYPE
        is
    variable opRslt     : signed(data_width downto 0)             := (others => '0');
    variable output     : data_type                               := (others => '0');
265 begin
        opRslt  := resize(value1, opRslt'length) + resize(value2, opRslt'length);
        if value1(DATA_WIDTH-1) = '0' and value2(DATA_WIDTH-1) = '0'  then
            if (opRslt(data_width) /= opRslt(data_width-1)) then
                output := max;
270         else
                output := value1 + value2;
            end if;
        elsif value1(DATA_WIDTH-1) = '1' and value2(DATA_WIDTH-1) = '1' then
            if (opRslt(DATA_WIDTH) /= opRslt(data_width-1)) then
275             output := min;
            else
                output  := opRslt(DATA_WIDTH-1 downto 0);
            end if;
        else
280         output  := opRslt(DATA_WIDTH-1 downto 0);
        end if;
        return output;
    end function;
    --==============================================
285 -- two 12Bits Subtraction
    --==============================================
    function subtractionOf12Bit(value1    : DATA_TYPE; value2       : DATA_TYPE)  return DATA_TYPE
        is
    variable opRslt                       : signed(data_width downto 0)        := (others => '0')
        ;
    variable output                       : data_type                          := (others => '0')
        ;
290 variable rslt                         : data_type                          := (others => '0')
        ;
    variable temp                         : data_type                          := (others => '0')
        ;
    begin
       if value1(DATA_WIDTH-1) = '0' and value2(DATA_WIDTH-1) = '0'  then
            output := value1 - value2;
295    elsif value1(DATA_WIDTH-1) = '0' and value2(DATA_WIDTH-1) = '1' then
            temp := twoComplement(value2);
            rslt := additionOf12Bit(value1, temp);
            if rslt(DATA_WIDTH-1) = '1' then
                output := max;
300         else
                output := rslt;
            end if;
       elsif value1(DATA_WIDTH-1) = '1' and value2(DATA_WIDTH-1) = '0' then
            temp := twoComplement(value2);
305         opRslt  := resize(value1, opRslt'length) + resize(temp, opRslt'length);
            if (opRslt(DATA_WIDTH) /= opRslt(DATA_WIDTH-1)) then  -- \"Uberlauf
```

```
              output := min;
          else
               rslt := additionOf12Bit(value1, temp);
310            output := rslt;
          end if;
    else
          temp := twoComplement(value2);
          rslt := additionOf12Bit(value1, temp);
315       output := rslt;
    end if;
     return output;
end function;
--==================================
320 -- two 12Bits Subtraction
--==================================
function return2LastCol(arg          : ADDR_TYPE)                    return ADDR_TYPE
      is
variable rslt          : ADDR_TYPE := (others => '0');
begin
325  rslt              := std_logic_vector(unsigned(arg)+21);
     return rslt;
end function;
--==================================
-- SIGN EXTENSION
330 --==================================
function signExtension(arg          : DATA_TYPE)                    return
     signEXT is
variable rslt       : signEXT       := (others => '0');
begin
     rslt                           := signEXT(arg(DATA_WIDTH-1)&'0'&arg(DATA_WIDTH-2 downto 0))
       ;
335  return rslt;
end function;
end DFT_FUNCTIONS;
```

Quellcode A.3: FUNCTIONS_PKG.vhd: Beschreibung allgemeiner Funktionen wie z.B. Addition oder Multiplikation.

```
--------------------------------------------------------------------------------
-- Company: Diese Entity ist ein Teil-Komponent des 2D-DFT Modul
-- ist sowohl in 1D-DFT als auch invertiert i 2D-DFT eingesetzt
-- berechnet die 210 Speicherzelle des BRAMs.
5 -- Engineer:
--
-- Create Date: 10/04/2018 04:38:59 PM
-- Design Name: Ada Koundoul Mastermikroelektronik
-- Module Name: Package - Behavioral
10 -- Project Name: Masterarbeit ISAR
-- Target Devices: Zedboard Xilinx
-- Tool Versions:
-- Description: Package Komponent
--
15 -- Dependencies:
--
-- Revision: Hamburg 24.10.2018
-- Revision 0.01 - File Created
-- Additional Comments:
20 --
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.Numeric_std.all;
25 use work.DSP_PKG.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
30
-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
35 package FUNCTIONS_PKG is
--==========================================
--************* FUNKTIONS_PROTOTYPE *************
--==========================================
function CALC_INTERPOL(ARG_1          : DATA_TYPE; ARG_2   : DATA_TYPE)   return DATA_TYPE;
40 function INC_ADDR(ADDR             : ADDR_TYPE)                          return ADDR_TYPE;
function JUMP2CELL(ADDR               : ADDR_TYPE)                          return ADDR_TYPE;
function JUMP1CELL_BACK(ADDR          : ADDR_TYPE)                          return ADDR_TYPE;
function JUM2NEXTROW_ADDR(ADDR        : ADDR_TYPE)                          return ADDR_TYPE;
function WRITE_32_BITS_ON_RAM(ARG_REAL : DATA_TYPE; ARG_IMG: DATA_TYPE)     return SLV;
45 --
end FUNCTIONS_PKG;
```

```
--
package body FUNCTIONS_PKG is
--==========================================================================================
--%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%BILDET DIE SUMME ZWEIER 12 BITS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
--==========================================================================================
function CALC_INTERPOL(ARG_1   : DATA_TYPE; ARG_2   : DATA_TYPE) return DATA_TYPE is
variable TEMP   : signed(DATA_WIDTH downto 0);                          -- 13 BITS FOR
    SUM of TWO 12BITS_DATAVALUE
variable RSLT   : DATA_TYPE;
    begin
        TEMP    := resize(ARG_1, TEMP'length) + resize(ARG_2, TEMP'length);     -- OPERANDEN UM
            1 BIT ERWEITERN
        RSLT    := TEMP(DATA_WIDTH downto 1);                           -- LEFT BITSHIFT
            => /2
        return RSLT;
end function;
--==========================================================================================
--%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%INCRMENT ADDRESS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
--==========================================================================================
function INC_ADDR(ADDR : ADDR_TYPE) return ADDR_TYPE is
variable INT_ADDR     : ADDR_TYPE     := ST_RE_ADDR;        -- initialisieren "0000000000"
    begin
        INT_ADDR  := std_logic_vector(unsigned(ADDR) + 1);
        return INT_ADDR;
end function;
--==========================================================================================
--%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SOLL ZWEI SPEICHERZELLEN SPRINGEN
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
--==========================================================================================
function JUMP2CELL(ADDR       : ADDR_TYPE)                          return ADDR_TYPE is
variable INT_ADDR     : ADDR_TYPE     := ST_RE_ADDR;        -- initialisieren "0000000000"
    begin
        INT_ADDR  := std_logic_vector(unsigned(ADDR) + 2);
        return INT_ADDR;
end function;
--==========================================================================================
--      SOLL ZWEI SPEICHERZELLEN ZURUECKSPRINGEN
--==========================================================================================
function JUMP1CELL_BACK(ADDR      : ADDR_TYPE)                          return ADDR_TYPE is
variable INT_ADDR     : ADDR_TYPE     := ST_RE_ADDR;        -- initialisieren "0000000000"
    begin
        INT_ADDR  := std_logic_vector(unsigned(ADDR) - 1);
        return INT_ADDR;
end function;
--==========================================================================================
--%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SOLL ZUR NAECHSTEN_ROW_INT SPRINGEN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
--==========================================================================================
function JUM2NEXTROW_ADDR(ADDR      : ADDR_TYPE)                        return ADDR_TYPE is
variable RSLT         : ADDR_TYPE := (others => '0');
begin
    RSLT    := std_logic_vector(unsigned(ADDR)+ 16);        -- + 16 0f 32Bits
    return RSLT;
end function;
--==========================================================================================
--%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% WRITE 32 BITS ON RAM %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
--==========================================================================================
function WRITE_32_BITS_ON_RAM(ARG_REAL: DATA_TYPE; ARG_IMG: DATA_TYPE) return SLV is
variable DATA   : SLV := (others => '0');
begin
    DATA    := std_logic_vector(ARG_REAL)&NULL_4SLV&std_logic_vector(ARG_IMG)&NULL_4SLV;
    return DATA;
end function;
--
end FUNCTIONS_PKG;
```

Quellcode A.4: Operate_PKG.vhd: Funktionen für die 2D-DFT.

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 30.09.2018 17:41:00
-- Design Name:
-- Module Name: Operate_PKG - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
```

```
20  ---------------------------------------------------------------------------------

    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    use ieee.NUMERIC_STD.all;
25  use work.FUNCTIONS_PKG.all;
    use work.DFT_FUNCTIONS.ALL;
    use work.DSP_PKG.all;

    -- Uncomment the following library declaration if using
30  -- arithmetic functions with Signed or Unsigned values
    --use IEEE.NUMERIC_STD.ALL;

    -- Uncomment the following library declaration if instantiating
    -- any Xilinx leaf cells in this code.
35  --library UNISIM;
    --use UNISIM.VComponents.all;

    package Operate_PKG is
    function twoCompl(arg : data_type)                          return data_type;
40  function add(arg1 : data_type; arg2     : data_type)        return data_type;
    function meanf(arg1 : data_type; arg2   : data_type)        return data_type;
    function subtrac(arg1 : data_type; arg2 : data_type)        return data_type;
    function mult(arg1 : data_type; arg2    : data_type)        return data_type;
    function invert(arg1 : data_type)                           return data_type;
45  function signExt (arg : data_type)                          return sign_type;
    --
    constant max        : data_type                     := "011111111111";
    constant min        : data_type                     := "100000000000";
    end Operate_PKG;
50
    package body Operate_PKG is
    --================================================================================
    -- 2Complement()------------------------------------------------------------------
    --================================================================================
55  function twoCompl(arg : data_type)              return data_type is
        variable output     : data_type         := (others => '0');
    begin
        output  := not(arg) + 1;
        return output;
60  end function;
    --================================================================================
    -- add()--------------------------------------------------------------------------
    --================================================================================
    function add(arg1 : data_type; arg2 : data_type)        return data_type is
65      variable opRslt     : signed(data_width downto 0)   := (others => '0');
        variable output     : data_type         := (others => '0');
    begin
        opRslt  := resize(arg1, opRslt'length) + resize(arg2, opRslt'length);
        if arg1(11) = '0' and arg2(11) = '0'  then
70          if (opRslt(data_width) /= opRslt(data_width-1)) then
                output := max;
            else
                output := arg1 + arg2;
            end if;
75      elsif arg1(11) = '1' and arg2(11) = '1' then
            if (opRslt(data_width) /= opRslt(data_width-1)) then
                output := min;
            else
                output  := opRslt(data_width-1 downto 0);
80          end if;
        else
            output  := opRslt(data_width-1 downto 0);
        end if;
        return output;
85  end function;
    --================================================================================
    -- meanf()------------------------------------------------------------------------
    --================================================================================
    function meanf(arg1 : data_type; arg2   : data_type)        return data_type is
90      variable opRslt     : signed(data_width downto 0)   := (others => '0');
        variable output     : data_type         := (others => '0');
    begin
        opRslt  := resize(arg1, opRslt'length) + resize(arg2, opRslt'length);
        output  := opRslt(data_width downto 1);
95      return output;
    end function;
    --================================================================================
    -- subtrac()----------------------------------------------------------------------
    --================================================================================
100 function subtrac(arg1 : data_type; arg2 : data_type)        return data_type is
        variable opRslt     : signed(data_width downto 0)   := (others => '0');
        variable output     : data_type                     := (others => '0');
        variable rslt       : data_type                     := (others => '0');
        variable temp       : data_type                     := (others => '0');
105 begin
       if arg1(11) = '0' and arg2(11) = '0'  then
```

```
        output := arg1 - arg2;
    elsif arg1(11) = '0' and arg2(11) = '1' then
        temp := twoCompl(arg2);          -- two Complement
110     rslt := add(arg1, temp);
        if rslt(11) = '1' then
            output := max;
        else
            output := rslt;
115     end if;
    elsif arg1(11) = '1' and arg2(11) = '0' then
        temp := twoCompl(arg2);
        opRslt  := resize(arg1, opRslt'length) + resize(temp, opRslt'length);
        if (opRslt(data_width) /= opRslt(data_width-1)) then  -- \"Uberlauf
120         output := min;
        else
            rslt := add(arg1, temp);
            output := rslt;
        end if;
125 else
        temp := twoCompl(arg2);
        rslt := add(arg1, temp);
        output := rslt;
    end if;
130  return output;
end function;
--==========================================================================
-- mult()-------------------------------------------------------------------
--==========================================================================
135 function mult(arg1 : data_type; arg2    : data_type)        return data_type is
    -- die Multiplikation nur gultig wenn arg < 1
    variable opRslt         : pdc_type      := (others => '0');
    variable output         : data_type     := (others => '0');
begin
140 opRslt   := pdc_type(arg1*arg2);
    output   := data_type(opRslt(pdc_width-1)&opRslt(pdc_width-4 downto 10));
    return output;
end function;
--==========================================================================
145 -- invert()-----------------------------------------------------------------
--==========================================================================
function invert(arg1 : data_type)                       return data_type is
    --variable opRslt       : pdc_type                  := (others => '0');
    variable output     : data_type                 := (others => '0');
150 constant minusEins  : data_type                 := "110000000000";
begin
        output := mult(arg1, minusEins);
    return output;
end function;
155 --==========================================================================
-- Sign_Extention()---------------------------------------------------------
--==========================================================================
function signExt (arg : data_type)              return sign_type is
    variable output     : sign_type         := (others => '0');
160 begin
    output   := resize(arg, output'length);
    return output;
end function;
--==========================================================================
165 end Operate_PKG;
```

Quellcode A.5: toplevel.vhd: Verbindung von allen implementierten Modulen.

```
--------------------------------------------------------------------------------
-- Entitiy : TOPLEVEL
--------------------------------------------------------------------------------
-- Copyright 2018
5  -- Filename           : toplevel.vhd
-- Creation date      : 2018-04-26
-- Authors(s)         : Jannes Helck
-- Version            : 1.00
-- Description        : Top level
10 --------------------------------------------------------------------------------
-- File History :
-- Date          Version      Author                      Comment
-- 2018-04-26    1.00         J.Helck, A.Koundoul         Creation of file
--------------------------------------------------------------------------------
15
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

20 use work.BRAM_PKG.all;
use work.MEMORY_CONTROL_PKG.all;
use work.MODULE_CONTROL_PKG.all;
use work.INTERPOL2D_PKG.all;
```

```
use work.ATAN2_PKG.all;
use work.DD_FILTER_PKG.all;
use work.DD_DFT_PKG.all;
--use work.TEST_MODULE_PKG.all;

entity TOPLEVEL is
port (
     CLK              : in  std_logic;
     nRESET           : in  std_logic;
     CTRL_EXT_EN      : in  bit;
     MOD_SEL_EXT_EN   : in  bit;
     MOD_OUT          : out std_logic_vector(2 downto 0);
     MOD_RDY          : out std_logic;
     MOD_CLK_EN       : in  bit;
     DSEL             : in  std_logic_vector(2 downto 0);
     WE               : in  bit;
     DIN              : in  std_logic_vector(9 downto 0);
     DOUT             : out std_logic_vector(7 downto 0)

);
end TOPLEVEL;

architecture ARCH of TOPLEVEL is

signal RAM_EN        : std_logic;
signal MOD_SEL       : std_logic_vector(2 downto 0);
signal MOD_CLK       : std_logic;
signal RDY           : std_logic;
signal ADDR_BUS      : std_logic_vector(9 downto 0);
signal DIN_BUS       : std_logic_vector(31 downto 0);
signal DOUT_BUS      : std_logic_vector(31 downto 0);

signal WE_RAM        : std_logic;
signal RE_RAM        : std_logic;

signal WE_C1         : std_logic;
signal RE_C1         : std_logic;
signal WE_C4         : std_logic;
signal RE_C4         : std_logic;
signal WE_C5         : std_logic;
signal RE_C5         : std_logic;
signal WE_C6         : std_logic;
signal RE_C6         : std_logic;
signal WE_C7         : std_logic;
signal RE_C7         : std_logic;


begin

RAM_EN  <= '1';
MOD_OUT <= MOD_SEL;
MOD_RDY <= RDY;
MOD_CLK <= CLK when MOD_CLK_EN = '1' else '0';

WE_RAM <= WE_C1 or WE_C4 or WE_C5 or WE_C6 or WE_C7;
RE_RAM <= RE_C1 or RE_C4 or RE_C5 or RE_C6 or RE_C7;

-- Instantiation of Component : MEMORY_CONTROL
C1: MEMORY_CONTROL port map (
     CLK              => CLK,
     nRESET           => nRESET,
     CTRL_EXT_EN      => CTRL_EXT_EN,
     MOD_SEL_EXT_EN   => MOD_SEL_EXT_EN,
     MOD_SEL          => MOD_SEL,
     DIN              => DIN,
     DOUT             => DOUT,
     DSEL             => DSEL,
     WE_IN            => WE,
     WE               => WE_C1,
     RE               => RE_C1,
     ADDR_BUS         => ADDR_BUS,
     DIN_BUS          => DIN_BUS,
     DOUT_BUS         => DOUT_BUS
);

-- Instantiation of Component: BRAM
C2: BRAM port map (
     CLK              => CLK,
     nRESET           => nRESET,
     EN               => RAM_EN,
     WE               => WE_RAM,
     RE               => RE_RAM,
     ADDR             => ADDR_BUS,
     DIN              => DIN_BUS,
     DOUT             => DOUT_BUS
);

-- Instantiation of Component: MODULE_CONTROL
```

```
    C3: MODULE_CONTROL port map (
        CLK              => MOD_CLK,
        nRESET           => nRESET,
115     MOD_SEL_EXT_EN   => MOD_SEL_EXT_EN,
        MOD_SEL          => MOD_SEL,
        RDY              => RDY
    );

120 C4: INTERPOL2D port map(
        CLK              => MOD_CLK,
        nRESET           => nRESET,
        EN               => CTRL_EXT_EN,
        DIN              => DOUT_BUS,
125     MODLCTRL         => MOD_SEL,
        ADDR             => ADDR_BUS,
        WE               => WE_C4,
        RE               => RE_C4,
        MODL_READY       => RDY,
130     DOUT             => DIN_BUS
    );
    -- Instantiation of Component: ATAN2
    C5: ATAN2 port map (
        CLK              => MOD_CLK,
135     nRESET           => nRESET,
        MOD_SEL          => MOD_SEL,
        CTRL_EXT_EN      => CTRL_EXT_EN,
        RDY              => RDY,
        WE               => WE_C5,
140     RE               => RE_C5,
        ADDR_BUS         => ADDR_BUS,
        DIN_BUS          => DIN_BUS,
        DOUT_BUS         => DOUT_BUS
    );
145 --
    C6: DD_FILTER port map(
        CLK              => MOD_CLK,
        nRESET           => nRESET,
        EN               => CTRL_EXT_EN,
150     DIN              => DOUT_BUS,
        MODLCTRL         => MOD_SEL,
        ADDR             => ADDR_BUS,
        WE               => WE_C6,
        RE               => RE_C6,
155     MODL_READY       => RDY,
        DOUT             => DIN_BUS
    );
    --
    C7: DD_DFT port map(
160     CLK              => MOD_CLK,
        nRESET           => nRESET,
        EN               => CTRL_EXT_EN,
        DIN              => DOUT_BUS,
        MODLCTRL         => MOD_SEL,
165     ADDR             => ADDR_BUS,
        WE               => WE_C7,
        RE               => RE_C7,
        MODL_READY       => RDY,
        DOUT             => DIN_BUS
170 );
    -- Instantiation of Component: TEST_MODULE_DD_DFT_PKG
    --C5: TEST_MODULE port map (
    --      CLK              => MOD_CLK,
    --      nRESET           => nRESET,
175 --      CTRL_EXT_EN      => CTRL_EXT_EN,
    --      MOD_SEL          => MOD_SEL,
    --      RDY              => RDY,
    --      WE               => WE_C5,
    --      RE               => RE_C5,
180 --      ADDR_BUS         => ADDR_BUS,
    --      DIN_BUS          => DIN_BUS,
    --      DOUT_BUS         => DOUT_BUS
    --);

185 end ARCH;
```

Quellcode A.6: interpol_2d.vhd: Lineare Interpolation von einem 8 × 8-Array auf ein 15 × 15-Array.

```
    ----------------------------------------------------------------------------------
    -- Entitiy : INTERPOL2D / %%%%% Masterthesis HAW Hamburg %%%%%
    ----------------------------------------------------------------------------------
    -- Copyright 2018
5   -- Filename          : INTERPOL2D.vhd
    -- Creation date     : 2018-05-09
```

```
-- Authors(s)          : Ada Koundoul / Master Mikroelektronische Systeme
-- Version             : 1.00
-- Description         : Signalverarbeitung f\"ur ein magnetisches Sensor-Array
--                       als digitaler Chipenwurf
--------------------------------------------------------------------------------
-- File History :
-- Date           Version      Author        Comment
-- 2018-05-09     1.00         A.Koundoul    Creation of file
--------------------------------------------------------------------------------

------------------------------------ Package -----------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.Numeric_std.all;
use work.DSP_PKG.all;
--==============================================================================
--%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% INTERPOL2D_PKG %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
--==============================================================================
package INTERPOL2D_PKG is
component INTERPOL2D is
  Port (
    --  MODUL_INPUTS
    CLK          : in STD_LOGIC;      -- STEUERSIGNALE
    nRESET       : in STD_LOGIC;
    EN           : in bit;
    DIN          : in SLV;            -- 32 Bits INPUT
    MODLCTRL     : in MODUL;          -- 3 Bits MODULSELECT
    --  MODUL_OUTPUTS
    ADDR         : out ADDR_TYPE;     -- 10 BITS
    WE           : out STD_LOGIC;     -- WRITEENABLE
    RE           : out STD_LOGIC;     -- READENABLE
    MODL_READY   : out STD_LOGIC;     -- from MODUL_2_CONTROLLER
    DOUT         : out SLV            -- 32 Bits OUTPUT
  );
end component;
end INTERPOL2D_PKG;
--==============================================================================
--%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% end Package %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
--==============================================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.Numeric_std.all;
use work.DSP_PKG.all;
use work.INT2D_ROW_PKG.all;
use work.INT2D_COLUMN_PKG.all;
--
entity INTERPOL2D is
  Port (
    --  MODUL_INPUTS
    CLK          : in STD_LOGIC;      -- STEUERSIGNALE
    nRESET       : in STD_LOGIC;
    EN           : in bit;
    DIN          : in SLV;            -- 32 Bits INPUT
    MODLCTRL     : in MODUL;          -- 3 Bits MODULSELECT
    --  MODUL_OUTPUTS
    ADDR         : out ADDR_TYPE;     -- 10 BITS
    WE           : out STD_LOGIC;     -- WRITEENABLE
    RE           : out STD_LOGIC;     -- READENABLE
    MODL_READY   : out STD_LOGIC;     -- from MODUL_2_CONTROLLER
    DOUT         : out SLV            -- 32 Bits OUTPUT
  );
end INTERPOL2D;
--================================================
--%%%%%%%%%%%%%%%%ARCHITECTURE_OF_ENTITY%%%%%%%%%%%%%%%%
--================================================
architecture ARCH of INTERPOL2D is
--============================================
--%%%%%%%%%%%%%%%%SIGNALDEKLARATION%%%%%%%%%%%%%%%%
--============================================
signal ADDR_BUS, ADDR_BUS_COL                              : ADDR_TYPE;
signal WENABLE, RENABLE, WENABLE_COL, RENABLE_COL          : STD_LOGIC;
signal ROW_READY, READY                                    : STD_LOGIC;
signal DATA_OUT,DATA_OUT_COL,   DATA_IN                    : SLV;
signal SEL                                                 : STD_LOGIC;
--
signal DOUT_int               : SLV;
signal ADDR_int               : ADDR_TYPE;
signal WE_int, RE_int         : STD_LOGIC;

begin
--
SEL      <= ROW_READY;
--
--========================================
--%%%%%%%%%% COMPONENTENVERDRAHTUNG %%%%%%%%%%
--========================================
ROWINT : INT2D_ROW port map(
 CLK            =>  CLK,
```

```
95     nRESET         =>  nRESET,
       DIN            =>  DIN,
       MODLCTRL       =>  MODLCTRL,
       ADDR           =>  ADDR_BUS,         --%%%%%%%%% OUTPUTS %%%%%%%%%
       WE             =>  WENABLE,
100    RE             =>  RENABLE,
       ROW_INT_DONE   =>  ROW_READY,
       DOUT           =>  DATA_OUT
     );
     -- --
105  COLUMNINT  : INT2D_COLUMN port map(
       CLK            =>  CLK,
       nRESET         =>  nRESET,
       DIN            =>  DIN,
       ROW_INT_DONE   =>  ROW_READY,
110    ADDR           =>  ADDR_BUS_COL,         --%%%%%%%%% OUTPUTS %%%%%%%%%
       WE             =>  WENABLE_COL,
       RE             =>  RENABLE_COL,
       MODL_READY     =>  READY,
       DOUT           =>  DATA_OUT_COL
115  );
     -- --

     DOUT_int     <= DATA_OUT_COL when    SEL ='1' else DATA_OUT;
     ADDR_int     <= ADDR_BUS_COL when    SEL ='1' else ADDR_BUS;
120  WE_int       <= WENABLE_COL  when    SEL ='1' else WENABLE;
     RE_int       <= RENABLE_COL  when    SEL ='1' else RENABLE;
     --==================================================
     --%%%%%%%%%% OUTPUTS INTERPOLATION %%%%%%%%%%
     --==================================================
125  -- Three state bus system drivers
     RE           <= RE_int   when MODLCTRL = INTMOD and EN = '0' else '0';
     WE           <= WE_int   when MODLCTRL = INTMOD and EN = '0' else '0';
     ADDR         <= ADDR_int when MODLCTRL = INTMOD and EN = '0' else (others=>'Z');
     DOUT         <= DOUT_int when MODLCTRL = INTMOD and EN = '0' else (others=>'Z');
130  MODL_READY   <= READY    when MODLCTRL = INTMOD else 'Z';
     end ARCH;
```

Quellcode A.7: INT2D_ROW.vhd: Interpolation der Reihen.

```
     --------------------------------------------------------------------------------
     -- Entitiy : INTERPOL2D / %%%%% Masterthesis HAW Hamburg %%%%%
     --------------------------------------------------------------------------------
     -- Copyright 2018
5    -- Filename          : INTERPOL2D.vhd
     -- Creation date     : 2018-05-09
     -- Authors(s)        : Ada Koundoul / Master Mikroelektronische Systeme
     -- Version           : 1.00
     -- Description       : Signalverarbeitung f\"ur ein magnetisches Sensor-Array
10   --                     als digitaler Chipenwurf
     --------------------------------------------------------------------------------
     -- File History :
     -- Date          Version     Author      Comment
     -- 2018-05-09    1.00        A.Koundoul  Creation of file
15   --------------------------------------------------------------------------------

     ------------------------------------ Package ------------------------------------
     library IEEE;
     use IEEE.STD_LOGIC_1164.ALL;
20   use work.DSP_PKG.all;
     --================================================================================
     --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% end Package %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     --================================================================================
     package INT2D_ROW_PKG is
25   component INT2D_ROW is
       Port (
         --  MODUL_INPUTS
         CLK               : in STD_LOGIC;      -- STEUERSIGNALE
         nRESET            : in STD_LOGIC;
30       DIN               : in SLV;            -- 32 Bits INPUT
         MODLCTRL          : in MODUL;          -- 3 Bits MODULSELECT
         --  MODUL_OUTPUTS
         ADDR              : out ADDR_TYPE;     -- 10 BITS
         WE                : out STD_LOGIC;     -- WRITEENABLE
35       RE                : out STD_LOGIC;     -- READENABLE
         ROW_INT_DONE      : out STD_LOGIC;     -- from MODUL_2_CONTROLLER
         DOUT              : out SLV            -- 32 Bits OUTPUT
       );
     end component;
40   end package;
     --================================================================================
     --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% end Package %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     --================================================================================
     library IEEE;
45   use IEEE.STD_LOGIC_1164.ALL;
```

```
    use ieee.Numeric_std.all;
    use work.DSP_PKG.all;
    use work.FUNCTIONS_PKG.all;
    --
50  entity INT2D_ROW is
      Port (
        --  MODUL_INPUTS
        CLK               : in STD_LOGIC;       -- STEUERSIGNALE
        nRESET            : in STD_LOGIC;
55      DIN               : in SLV;             -- 32 Bits INPUT
        MODLCTRL          : in MODUL;           -- 3 Bits MODULSELECT
        --  MODUL_OUTPUTS
        ADDR              : out ADDR_TYPE;      -- 10 BITS ADDRESSE
        WE                : out STD_LOGIC;      -- WRITEENABLE
60      RE                : out STD_LOGIC;      -- READENABLE
        ROW_INT_DONE      : out STD_LOGIC;      -- from MODUL_2_CONTROLLER
        DOUT              : out SLV             -- 32 Bits OUTPUT
      );
    end INT2D_ROW;
65  --
    architecture Behavioral of INT2D_ROW is
    --**********SIGNALDEKLARATION**********
    signal QPLUS                 : QCOUNTER;      -- 4 BITS UNSIGNED COUNTER
    signal Q                     : QCOUNTER;
70
    signal ROW_INT_DONE_INTERN      :   std_logic    :=  '0';
    --
    begin
    --=====================================================================
75  --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SAVECOUNTER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    --=====================================================================
    COUNTER_P :   process(nRESET, CLK, MODLCTRL)
    begin
        if nRESET = '0' or MODLCTRL /= INTMOD then
80          Q <= QCTN0;
        elsif CLK'event and CLK = '1' then
            Q <= QPLUS;                          -- COUNT
        end if;
    end process;
85  --=====================================================================
    --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COUNTER_P %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    --=====================================================================
    COUNT_CLK : process(Q, nRESET, MODLCTRL)
    variable QEND    : QCOUNTER  := QCTN24;         -- 14 Takte
90  begin

        if nRESET = '0' or MODLCTRL /= INTMOD then
            QPLUS    <= QCTN0;
        elsif Q = QEND then
95          if ROW_INT_DONE_INTERN = '0' then
                QPLUS    <= QCTN1;                      -- QPLUS ZURUECKSETZEN
            end if;
         else
            if ROW_INT_DONE_INTERN = '0' then
100         QPLUS    <= Q + 1;                          -- count CLK'EVENT
            end if;
        end if;
    end process;
    --=====================================================================
105 --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% OPERATING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    --=====================================================================
    DATA_RAM : process(QPLUS, CLK)
    -- ADDR_SETTINGS
    variable RE_ADDR                               : ADDR_TYPE := (others => '0');
110 variable WR_ADDR                               : ADDR_TYPE := ST_WE_ADDR;
    variable TMP_ADDR                              : ADDR_TYPE := (others => '0');
    --
    variable realroh_0, imgroh_0                   :  data_type   := (others => '0');
    variable realroh_1, imgroh_1                   :  data_type   := (others => '0');
115 variable real_Interpol,  img_interpol          :  data_type   := (others => '0');
    --
    begin
        if CLK'event and CLK = '0' then          -- FALLING_EDGE(CLK)
            case QPLUS is
120             when QCTN0  =>    -- if RESET
                                                                                    --
                    ===================================
                    ADDR                <= (others => '0');
                                                                              --
                        INITIALISIERUNG BEI RESET
                    WE                  <= '0'; RE <= '0';
                                                                               -- wird
                        weder geschrieben noch gelesen
                    ROW_INT_DONE        <= '0';
                    DOUT                <= (others =>'0');
125                 RE_ADDR             := (others => '0');
                                                                              -- RE_ADDR
                        beibehalten
```

```
          WR_ADDR              := ST_WE_ADDR;                                  --
               WR_ADDR beibehalten
      when QCTN1 =>
           -- ==========================================
          ADDR                 <=  RE_ADDR;
          RE                   <= HIGHLEVEL;
130   when QCTN2 =>
           -- ==========================================
          realroh_0            := signed(DIN(R_MSB downto R_LSB));
          imgroh_0             := signed(DIN(I_MSB downto I_LSB));
          RE_ADDR              := INC_ADDR(RE_ADDR);
          ADDR                 <=  RE_ADDR;
135   when QCTN3 =>
          realroh_1            := signed(DIN(R_MSB downto R_LSB));
          imgroh_1             := signed(DIN(I_MSB downto I_LSB));
          RE                   <= LOWLEVEL;

140       RE_ADDR              := INC_ADDR(RE_ADDR);                           --
               inc(RE_ADDR) and wait for next READ
          real_Interpol        := CALC_INTERPOL(realroh_0, realroh_1);
          img_Interpol         := CALC_INTERPOL(imgroh_0, imgroh_1);

          ADDR                 <= WR_ADDR;
145       WE                   <= HIGHLEVEL;                                   --
               we high
          DOUT                 <= WRITE_32_BITS_ON_RAM(realroh_0, imgroh_0);
                    -- WRITE ROHDATA ADDR 64
      when QCTN4 =>
           -- ==========================================
          WR_ADDR              := INC_ADDR(WR_ADDR);                           --
               inc(WR_ADDR)
          ADDR                 <=  WR_ADDR;
150       DOUT                 <= WRITE_32_BITS_ON_RAM(real_Interpol, img_Interpol);
                              -- WRITE_ACTION 32 BITS INTERPOL DATA
      when QCTN5 =>
          WR_ADDR              :=INC_ADDR(WR_ADDR);                            --
               inc(WR_ADDR)
          ADDR                 <=  WR_ADDR;
          DOUT                 <= WRITE_32_BITS_ON_RAM(realroh_1, imgroh_1);
155   when QCTN6 =>
          WE                   <= LOWLEVEL;
          ADDR                 <=  RE_ADDR;
          RE                   <= HIGHLEVEL;
      when QCTN7 =>
160       realroh_0            :=  signed(DIN(R_MSB downto R_LSB));
          imgroh_0             :=  signed(DIN(I_MSB downto I_LSB));
          RE_ADDR              :=INC_ADDR(RE_ADDR);
          RE                   <= LOWLEVEL;

165       real_Interpol        := CALC_INTERPOL(realroh_0, realroh_1);
          img_Interpol         := CALC_INTERPOL(imgroh_0, imgroh_1);
          WR_ADDR              :=INC_ADDR(WR_ADDR);                            --
               inc(WR_ADDR)
          WE                   <= HIGHLEVEL;                                   --
               we high
          ADDR                 <=  WR_ADDR;
170       DOUT                 <= WRITE_32_BITS_ON_RAM(real_Interpol, img_Interpol);

      when QCTN8 =>
          WR_ADDR              :=INC_ADDR(WR_ADDR);                            --
               inc(WR_ADDR)
          ADDR                 <=  WR_ADDR;
175       DOUT                 <= WRITE_32_BITS_ON_RAM(realroh_0, imgroh_0);

      when QCTN9 =>
         WE                   <= LOWLEVEL;
         ADDR                 <=  RE_ADDR;
180      RE                   <= HIGHLEVEL;
      when QCTN10 =>
          realroh_1            :=  signed(DIN(R_MSB downto R_LSB));
          imgroh_1             :=  signed(DIN(I_MSB downto I_LSB));
          RE_ADDR              :=INC_ADDR(RE_ADDR);                            --
               inc(RE_ADDR) and wait for next READ
185       RE                   <= LOWLEVEL;

          real_Interpol        := CALC_INTERPOL(realroh_0, realroh_1);
          img_Interpol         := CALC_INTERPOL(imgroh_0, imgroh_1);

190       WR_ADDR              :=INC_ADDR(WR_ADDR);                            --
               inc(WR_ADDR)
          WE                   <= HIGHLEVEL;
          ADDR                  <=  WR_ADDR;
          DOUT                  <= WRITE_32_BITS_ON_RAM(real_Interpol, img_Interpol);
      when QCTN11 =>
```

```
195                 WR_ADDR                 :=INC_ADDR(WR_ADDR);                                    --
                        inc(WR_ADDR)
                    ADDR                    <=  WR_ADDR;
                    DOUT                    <= WRITE_32_BITS_ON_RAM(realroh_1, imgroh_1);
                when QCTN12 =>

                    -- ==================================================
                    WE                      <= LOWLEVEL;
200                 ADDR                    <=  RE_ADDR;
                    RE                      <= HIGHLEVEL;
                when QCTN13 =>
                    realroh_0               :=  signed(DIN(R_MSB downto R_LSB));
                    imgroh_0                :=  signed(DIN(I_MSB downto I_LSB));
205                 RE_ADDR                 := INC_ADDR(RE_ADDR);
                    RE                      <= LOWLEVEL;

                    real_Interpol           := CALC_INTERPOL(realroh_0, realroh_1);
                    img_Interpol            := CALC_INTERPOL(imgroh_0, imgroh_1);
210
                    WR_ADDR                 :=INC_ADDR(WR_ADDR);                                    --
                        inc(WR_ADDR)
                    ADDR                   <=  WR_ADDR;
                    WE                      <= HIGHLEVEL;
                    DOUT                   <= WRITE_32_BITS_ON_RAM(real_Interpol, img_Interpol);
215             when QCTN14 =>
                    WR_ADDR                :=INC_ADDR(WR_ADDR);                                     --
                        inc(WR_ADDR)
                    ADDR                   <=  WR_ADDR;
                    DOUT                   <= WRITE_32_BITS_ON_RAM(realroh_0, imgroh_0);
                when QCTN15 =>
220                 WE                     <= LOWLEVEL;
                    ADDR                   <=  RE_ADDR;
                    RE                     <= HIGHLEVEL;
                when QCTN16 =>

                    -- ==================================================
                    realroh_1              :=  signed(DIN(R_MSB downto R_LSB));
225                 imgroh_1               :=  signed(DIN(I_MSB downto I_LSB));
            RE_ADDR                 :=INC_ADDR(RE_ADDR);
            --
                    real_Interpol          := CALC_INTERPOL(realroh_0, realroh_1);
                    img_Interpol           := CALC_INTERPOL(imgroh_0, imgroh_1);
230
                    RE                     <= LOWLEVEL;
                    WR_ADDR                :=INC_ADDR(WR_ADDR);
                    ADDR                    <=  WR_ADDR;
                    WE                     <= HIGHLEVEL;
235                 DOUT                    <= WRITE_32_BITS_ON_RAM(real_Interpol, img_Interpol);
                when QCTN17 =>
                    WR_ADDR                 :=INC_ADDR(WR_ADDR);
                    ADDR                    <=  WR_ADDR;
                    DOUT                    <= WRITE_32_BITS_ON_RAM(realroh_1, imgroh_1);
240             when QCTN18 =>
                    WE                      <= LOWLEVEL;
                    ADDR                    <=  RE_ADDR;
                    RE                      <= HIGHLEVEL;
                when QCTN19 =>
245                 realroh_0                :=  signed(DIN(R_MSB downto R_LSB));
                    imgroh_0                 :=  signed(DIN(I_MSB downto I_LSB));
                    RE_ADDR                  :=INC_ADDR(RE_ADDR);
                --
                    real_Interpol            := CALC_INTERPOL(realroh_0, realroh_1);
250                 img_Interpol             := CALC_INTERPOL(imgroh_0, imgroh_1);

                    RE                      <= LOWLEVEL;
                    WR_ADDR                 :=INC_ADDR(WR_ADDR);
                    ADDR                    <=  WR_ADDR;
255                 WE                      <= HIGHLEVEL;
                    DOUT                    <= WRITE_32_BITS_ON_RAM(real_Interpol, img_Interpol);
                when QCTN20 =>
                    WR_ADDR                 :=INC_ADDR(WR_ADDR);
                    ADDR                    <=  WR_ADDR;
260                 DOUT                    <= WRITE_32_BITS_ON_RAM(realroh_0, imgroh_0);
                when QCTN21 =>
                    WE                      <= LOWLEVEL;
                    ADDR                    <=  RE_ADDR;
                    RE                      <= HIGHLEVEL;
265             when QCTN22 =>
                    realroh_1               :=  signed(DIN(R_MSB downto R_LSB));
                    imgroh_1                :=  signed(DIN(I_MSB downto I_LSB));
                    RE_ADDR                 :=  INC_ADDR(RE_ADDR);

270                 real_Interpol           := CALC_INTERPOL(realroh_0, realroh_1);
                    img_Interpol            := CALC_INTERPOL(imgroh_0, imgroh_1);

                    RE                      <= LOWLEVEL;
                    WR_ADDR                 :=INC_ADDR(WR_ADDR);
275                 ADDR                    <=  WR_ADDR;
```

```
                    WE                        <= HIGHLEVEL;
                    DOUT                      <= WRITE_32_BITS_ON_RAM(real_Interpol, img_Interpol);
                when QCTN23 =>
                    WE                        <= LOWLEVEL;
280                 WR_ADDR                   :=INC_ADDR(WR_ADDR);
                    ADDR                      <=  WR_ADDR;
                    WE                        <= HIGHLEVEL;
                    DOUT                      <= WRITE_32_BITS_ON_RAM(realroh_1, imgroh_1);
                when QCTN24 =>
285                 if ROW_INT_DONE_INTERN = '0' then
                        WE            <= LOWLEVEL;
                        RE            <= LOWLEVEL;
                        if to_integer(unsigned(RE_ADDR)) >= 63 then
                            ROW_INT_DONE <= '1';
290                         ROW_INT_DONE_INTERN <= '1';
                        else
                            WR_ADDR       :=JUM2NEXTROW_ADDR(WR_ADDR);
                        end if;
                    end if;
295             when others => null;
            end case;
        end if;
    end process;
    --
300 end Behavioral;
```

Quellcode A.8: INT2D_COLUMN.vhd: Interpolation der Spalten.

```
    -----------------------------------------------------------------------------
    -- Entitiy : INTERPOL2D / %%%%% Masterthesis HAW Hamburg %%%%%
    -----------------------------------------------------------------------------
    -- Copyright 2018
5   -- Filename           : INTERPOL2D.vhd
    -- Creation date      : 2018-05-20
    -- Authors(s)         : Ada Koundoul / Master Mikroelektronische Systeme
    -- Version            : 1.00
    -- Description        : Signalverarbeitung f\"ur ein magnetisches Sensor-Array
10  --                      als digitaler Chipenwurf
    -----------------------------------------------------------------------------
    -- File History :
    -- Date           Version      Author       Comment
    -- 2018-05-09     1.00         A.Koundoul   Creation of file
15  -----------------------------------------------------------------------------

    ----------------------------------- Package ----------------------------------
    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
20  use work.DSP_PKG.all;
    --==========================================================================
    --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PACKAGE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    --==========================================================================
    package INT2D_COLUMN_PKG is
25  component INT2D_COLUMN is
      Port (
        --  MODUL_INPUTS
        CLK               : in STD_LOGIC;      -- STEUERSIGNALE
        nRESET            : in STD_LOGIC;
30      DIN               : in SLV;            -- 32 Bits INPUT
        ROW_INT_DONE      : in STD_LOGIC;     -- from MODUL_2_CONTROLLER
        --  MODUL_OUTPUTS
        ADDR              : out ADDR_TYPE;     -- 10 BITS
        WE                : out STD_LOGIC;     -- WRITEENABLE
35      RE                : out STD_LOGIC;     -- READENABLE
        MODL_READY        : out STD_LOGIC;     -- from MODUL_2_CONTROLLER
        DOUT              : out SLV            -- 32 Bits OUTPUT
      );
    end component;
40  end package;
    --==========================================================================
    --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PACKAGE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    --==========================================================================
    library IEEE;
45  use IEEE.STD_LOGIC_1164.ALL;
    use ieee.Numeric_std.all;
    use work.DSP_PKG.all;
    use work.FUNCTIONS_PKG.all;
    --
50  entity INT2D_COLUMN is
      Port (
        --  MODUL_INPUTS
        CLK               : in STD_LOGIC;      -- STEUERSIGNALE
        nRESET            : in STD_LOGIC;
55      DIN               : in SLV;            -- 32 Bits INPUT
        ROW_INT_DONE      : in STD_LOGIC;     -- from MODUL_2_CONTROLLER
        --  MODUL_OUTPUTS
```

```
        ADDR            : out ADDR_TYPE;       -- 10 BITS
        WE              : out STD_LOGIC;       -- WRITEENABLE
60      RE              : out STD_LOGIC;       -- READENABLE
        MODL_READY      : out STD_LOGIC;       -- from MODUL_2_CONTROLLER
        DOUT            : out SLV              -- 32 Bits OUTPUT
    );
end INT2D_COLUMN;
65  --
architecture Behavioral of INT2D_COLUMN is
--
signal QPLUS                  : QCOUNTER;      -- 4 BITS UNSIGNED COUNTER
signal Q                      : QCOUNTER;
70  --
signal col_INT_DONE_INTERN    :  std_logic   := '0';
begin
--==========================================================================
--%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SAVECOUNT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
75  --==========================================================================
SAVE_COUNT : process(nRESET, CLK, ROW_INT_DONE)
begin
    if nRESET = '0' or ROW_INT_DONE = '0'then
        Q <= QCTN0;
80      elsif CLK'event and CLK = '1' then
        Q <= QPLUS;
    end if;
end process;
--==========================================================================
85  --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COUNT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
--==========================================================================
COUNT_CLK : process(Q, nRESET, ROW_INT_DONE)
variable QEND   : QCOUNTER  := QCTN34;       -- 34 Takte fuer eine Zeile
begin
90      if nRESET = '0' or ROW_INT_DONE = '0'then
        QPLUS    <= QCTN0;
    elsif Q = QEND then
        if col_INT_DONE_INTERN = '0' then
            QPLUS   <= QCTN1;                    -- QPLUS ZURUECKSETZEN
95          end if;
     else
        if col_INT_DONE_INTERN = '0' then
            QPLUS   <= Q + 1;                        -- count CLK'EVENT
        end if;
100     end if;
end process;
--==========================================================================
--%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% OPERATING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
--==========================================================================
105 DATA_RAM : process(QPLUS, CLK)
--
variable RE_ADDR0                         : ADDR_TYPE            := ST_WE_ADDR;
    -- ab dieser ADDR f\"ur READ0
variable RE_ADDR1                         : ADDR_TYPE            := ST_ADD_READ1;
    -- ab dieser ADDR f\"ur READ1
variable WR_ADDR                          : ADDR_TYPE            := ST_ADDR_WRITE;
    -- ab dieser ADDR f\"ur WRITE
110 --
variable roh_real0, roh_img0              :  data_type           := (others => '0');
variable roh_real1, roh_img1              :  data_type           := (others => '0');
variable interp_real,  interp_img         :  data_type           := (others => '0');
--
115 variable currentREAL0, currentIMG0        : data_type             := (others => '0');
                    -- store 12Bits  REAL & IMG  from READ1
variable currentREAL1, currentIMG1        : data_type             := (others => '0');
variable ROUND                            : unsigned(5 downto 0)  := (others => '0');
                    -- 6 Bits count
begin
    if CLK'event and CLK = '0' then          -- FALLING_EDGE(CLK)
120         case QPLUS is
            when QCTN0 =>

                -- INIT AND RESET
                ADDR        <= (others =>'0');
                DOUT        <= (others =>'0');
                WE          <= '0';
125             RE          <= '0';
                MODL_READY  <= '0';
                --
                RE_ADDR0            := ST_WE_ADDR;        -- start addr0 64
                RE_ADDR1            := ST_ADD_READ1;      -- start addr1 94
130             WR_ADDR             := ST_ADDR_WRITE;     -- start write_addr 79
            when QCTN1 =>

                -- ========================================
                ADDR                <= RE_ADDR0;
                                                                  -- ADDR anlegen,
                    erste Zeile
                RE                  <= HIGHLEVEL;
                                                                -- Read_Enable
```

```
            aktivieren
        when QCTN2  =>                          -- READ1
                                                        --
            ================================================
135         roh_real0           := signed(DIN(R_MSB downto R_LSB));
                                           -- READ1, real0 aus der ersten Zeile
            roh_img0            := signed(DIN(I_MSB downto I_LSB));
                                           -- img0       aus der ersten Zeile
            ADDR                <= RE_ADDR1;
                                                              -- READ1 for next
                Read in ROW2 anlegen
        when QCTN3 =>

            -- ================================================
            roh_real1           := signed(DIN(R_MSB downto R_LSB));
                                           -- READ1, real0 aus der ersten Zeile
140         roh_img1            := signed(DIN(I_MSB downto I_LSB));
                                           -- img0       aus der ersten Zeile
            interp_real         := CALC_INTERPOL(roh_real0, roh_real1);
                                       -- Vertikale Interpolation wird berechnet
            interp_img          := CALC_INTERPOL(roh_img0,  roh_img1);
            currentREAL0        := interp_real;
                                                           -- store Value for next
                  interpol
            currentIMG0         := interp_img;
145         --
            RE                  <= LOWLEVEL;
                                                              -- NO DATA READING
            RE_ADDR0            := JUMP2CELL(RE_ADDR0);
                                                  -- increment ADDR0, jump to
                next value on row1
            RE_ADDR1            := JUMP2CELL(RE_ADDR1);
                                                  -- JUMP2x READ1
            --
150         ADDR                <= WR_ADDR;
                                                               -- WRITE_ADDR
                anlegen
            WE                  <= HIGHLEVEL;
                                                                --
            DOUT                <= WRITE_32_BITS_ON_RAM(interp_real, interp_img);
                                  -- write intData First Write
        when QCTN4  =>
                                                                          --
              ================================================
            WE                  <= LOWLEVEL;
                                                              -- WE not aktiv
155         ADDR                <= RE_ADDR0;
                                                              -- Zeiger auf
                SpeicherZELLE(ROW0)
            RE                  <= HIGHLEVEL;
                                                             -- READ1 aktiv
        when QCTN5 =>                           -- READ2
                                                        --
            ================================================
            roh_real0           := signed(DIN(R_MSB downto R_LSB));
                                           -- READ0, real0 aus der ersten Zeile
            roh_img0            := signed(DIN(I_MSB downto I_LSB));
                                           -- img0       aus der ersten Zeile
160         ADDR                <= RE_ADDR1;
                                                              -- WE aktiv
        when QCTN6 =>

            -- ================================================
            roh_real1           := signed(DIN(R_MSB downto R_LSB));
                                           -- READ0, real0 aus der ersten Zeile
            roh_img1            := signed(DIN(I_MSB downto I_LSB));
                                           -- img0       aus der ersten Zeile
            interp_real         := CALC_INTERPOL(roh_real0, roh_real1);
                                       -- Vertikale Interpolation wird berechnet
165         interp_img          := CALC_INTERPOL(roh_img0,  roh_img1);
            currentREAL1        := interp_real;
                                                           -- store Value for next
                  interpol
            currentIMG1         := interp_img;

            RE                  <= LOWLEVEL;
                                                              -- NO DATA READING
170         RE_ADDR0            := JUMP2CELL(RE_ADDR0);
                                                  -- increment ADDR0, jump to
                next value on row1
            RE_ADDR1            := JUMP2CELL(RE_ADDR1);
                                                  -- JUMP 2x READ1

            WR_ADDR             := JUMP2CELL(WR_ADDR);
                                                   -- jum 2 Ze
            ADDR                <= WR_ADDR;
175         WE                  <= HIGHLEVEL;
            DOUT                <= WRITE_32_BITS_ON_RAM(interp_real, interp_img);
```

```
            when QCTN7 =>
                interp_real         := CALC_INTERPOL(currentREAL1, currentREAL0);
                                            -- Vertikale Interpolation wird berechnet
                interp_img          := CALC_INTERPOL(currentIMG1,  currentIMG0);
180
                WR_ADDR             :=JUMP1CELL_BACK(WR_ADDR);
                                                                    -- Addr zurueckstellen und write
                ADDR                <= WR_ADDR;
                                                                                                --
                    ==========================================================
                DOUT                <= WRITE_32_BITS_ON_RAM(interp_real, interp_img);
                                       -- 1 KOMPLET
            when QCTN8 =>
185             WR_ADDR             :=INC_ADDR(WR_ADDR);
                                                                        -- POSITIONIEREN
                                                                                                --
                    ==========================================================
                WE                  <= LOWLEVEL;
                                                                                -- Zeiger auf
                    SpeicherZELLE(ROW0)
                ADDR                <= RE_ADDR0;
                                                                                -- Zeiger auf
                    SpeicherZELLE(ROW0)
                RE                  <= HIGHLEVEL;
                                                                                -- READ1 aktiv
            when QCTN9 =>       -- READ3
                                                                                            --
                ==========================================================
190             roh_real0           := signed(DIN(R_MSB downto R_LSB));
                                                         -- READ0, real0 aus der ersten Zeile
                roh_img0            := signed(DIN(I_MSB downto I_LSB));
                ADDR                <= RE_ADDR1;
            when QCTN10 =>
                roh_real1           := signed(DIN(R_MSB downto R_LSB));
                                                         -- READ0, real0 aus der ersten Zeile
195             roh_img1            := signed(DIN(I_MSB downto I_LSB));
                                                         --
                    ==========================================================
                interp_real         := CALC_INTERPOL(roh_real0, roh_real1);
                                                     -- Vertikale Interpolation wird berechnet
                interp_img          := CALC_INTERPOL(roh_img0,  roh_img1);
                currentREAL0        := interp_real;
                                                                                -- store Value for next
                    interpol
                currentIMG0         := interp_img;
200             --
                RE                  <= LOWLEVEL;
                                                                                -- RE LOW
                RE_ADDR0            := JUMP2CELL(RE_ADDR0);
                                                                    -- increment ADDR0, jump to
                    next value on row1
                RE_ADDR1            := JUMP2CELL(RE_ADDR1);
                                                                    -- JUMP 2x READ1

205             WR_ADDR             := JUMP2CELL(WR_ADDR);
                ADDR                <= WR_ADDR;
                WE                  <= HIGHLEVEL;
                DOUT                <= WRITE_32_BITS_ON_RAM(interp_real, interp_img);
            when QCTN11 =>
210             interp_real         := CALC_INTERPOL(currentREAL1, currentREAL0);
                                            -- Vertikale Interpolation wird berechnet
                interp_img          := CALC_INTERPOL(currentIMG1,  currentIMG0);

                WR_ADDR             :=JUMP1CELL_BACK(WR_ADDR);
                ADDR                <= WR_ADDR;
                                                                                                --
                    ==========================================================
215             DOUT                <= WRITE_32_BITS_ON_RAM(interp_real, interp_img);
            when QCTN12 =>
                WR_ADDR             :=INC_ADDR(WR_ADDR);     -- POSITIONIEREN
                WE                  <= LOWLEVEL;
                                                                                -- Zeiger auf
                    SpeicherZELLE(ROW0)
                ADDR                <= RE_ADDR0;
                                                                                -- Zeiger auf
                    SpeicherZELLE(ROW0)
220             RE                  <= HIGHLEVEL;
            when QCTN13 =>

                 -- READ4
                roh_real0           := signed(DIN(R_MSB downto R_LSB));
                                                         -- READ0, real0 aus der ersten Zeile
                roh_img0            := signed(DIN(I_MSB downto I_LSB));
                ADDR                <= RE_ADDR1;
225         when QCTN14 =>
                roh_real1           := signed(DIN(R_MSB downto R_LSB));
                                                         -- READ0, real0 aus der ersten Zeile
                roh_img1            := signed(DIN(I_MSB downto I_LSB));
```

```
                                                                                   --
                 ================================================
            interp_real          := CALC_INTERPOL(roh_real0 , roh_real1);
                                                 -- Vertikale Interpolation wird berechnet
            interp_img           := CALC_INTERPOL(roh_img0 ,  roh_img1);
230         currentREAL1         := interp_real;
                                                                      -- store Value for
                 next interpol
            currentIMG1          := interp_img;

            RE                   <= LOWLEVEL;
                                                                           -- RE LOW
            RE_ADDR0             := JUMP2CELL(RE_ADDR0);
                                                           -- increment ADDR0, jump to
                 next value on row1
235         RE_ADDR1             := JUMP2CELL(RE_ADDR1);
                                                           -- JUMP 2x READ1

            WR_ADDR              := JUMP2CELL(WR_ADDR);
            ADDR                 <= WR_ADDR;
            WE                   <= HIGHLEVEL;
240         DOUT                 <= WRITE_32_BITS_ON_RAM(interp_real , interp_img);
        when QCTN15 =>
            interp_real          := CALC_INTERPOL(currentREAL1 , currentREAL0);
                                              -- Vertikale Interpolation wird berechnet
            interp_img           := CALC_INTERPOL(currentIMG1 ,  currentIMG0);

245         WR_ADDR              :=JUMP1CELL_BACK(WR_ADDR);
            ADDR                 <= WR_ADDR;
                                                                               --
                 ================================================
            DOUT                 <= WRITE_32_BITS_ON_RAM(interp_real , interp_img);
        when QCTN16 =>
            WR_ADDR              :=INC_ADDR(WR_ADDR);   -- POSITIONIEREN
250         WE                   <= LOWLEVEL;
                                                                           -- Zeiger auf
                 SpeicherZELLE(ROW0)
            ADDR                 <= RE_ADDR0;
                                                                           -- Zeiger auf
                 SpeicherZELLE(ROW0)
            RE                   <= HIGHLEVEL;
        when QCTN17 =>

            -- READ5
            roh_real0            := signed(DIN(R_MSB downto R_LSB));
                                                      -- READ0, real0 aus der ersten Zeile
255         roh_img0             := signed(DIN(I_MSB downto I_LSB));
            ADDR                 <= RE_ADDR1;
        when QCTN18 =>
            roh_real1            := signed(DIN(R_MSB downto R_LSB));
                                                      -- READ0, real0 aus der ersten Zeile
            roh_img1             := signed(DIN(I_MSB downto I_LSB));
                                                      --
                 ================================================
260         interp_real          := CALC_INTERPOL(roh_real0 , roh_real1);
                                                 -- Vertikale Interpolation wird berechnet
            interp_img           := CALC_INTERPOL(roh_img0 ,  roh_img1);
            currentREAL0         := interp_real;
                                                                      -- store Value for
                  next interpol
            currentIMG0          := interp_img;

265         RE                   <= LOWLEVEL;
                                                                           -- RE LOW
            RE_ADDR0             := JUMP2CELL(RE_ADDR0);
                                                           -- increment ADDR0, jump
                 to next value on row1
            RE_ADDR1             := JUMP2CELL(RE_ADDR1);
                                                           -- JUMP 2x READ1

            WR_ADDR              := JUMP2CELL(WR_ADDR);
270         ADDR                 <= WR_ADDR;
            WE                   <= HIGHLEVEL;
            DOUT                 <= WRITE_32_BITS_ON_RAM(interp_real , interp_img);
        when QCTN19 =>
            interp_real          := CALC_INTERPOL(currentREAL0 , currentREAL1);
                                              -- Vertikale Interpolation wird berechnet
275         interp_img           := CALC_INTERPOL(currentIMG0 ,  currentIMG1);

            WR_ADDR              :=JUMP1CELL_BACK(WR_ADDR);
            ADDR                 <= WR_ADDR;
                                                                               --
                 ================================================
            DOUT                 <= WRITE_32_BITS_ON_RAM(interp_real , interp_img);
280     when QCTN20 =>
            WR_ADDR              :=INC_ADDR(WR_ADDR);   -- POSITIONIEREN
            WE                   <= LOWLEVEL;
            ADDR                 <= RE_ADDR0;
```

```
                RE                      <= HIGHLEVEL;
285         when QCTN21 =>          -- READ6
                roh_real0               := signed(DIN(R_MSB downto R_LSB));
                                                                        -- READ0, real0 aus der ersten
                     Zeile
                roh_img0                := signed(DIN(I_MSB downto I_LSB));
                ADDR                    <= RE_ADDR1;
            when QCTN22 =>
290             roh_real1               := signed(DIN(R_MSB downto R_LSB));
                                                                        -- READ0, real0 aus der ersten
                     Zeile
                roh_img1                := signed(DIN(I_MSB downto I_LSB));
                                                                                                    --
                     ===============================================
                interp_real             := CALC_INTERPOL(roh_real0, roh_real1);
                                                                    -- Vertikale Interpolation wird
                     berechnet
                interp_img              := CALC_INTERPOL(roh_img0,  roh_img1);
                currentREAL1            := interp_real;
                                                                                                    -- store
                     Value for next interpol
295             currentIMG1             := interp_img;

                RE                      <= LOWLEVEL;
                                                                                                    -- RE
                     LOW
                RE_ADDR0                := JUMP2CELL(RE_ADDR0);
                                                                                    -- increment ADDR0,
                      jump to next value on row1
                RE_ADDR1                := JUMP2CELL(RE_ADDR1);
                                                                                    -- JUMP 2x READ1
300
                WR_ADDR                 := JUMP2CELL(WR_ADDR);
                ADDR                    <= WR_ADDR;
                WE                      <= HIGHLEVEL;
                DOUT                    <= WRITE_32_BITS_ON_RAM(interp_real, interp_img);
305         when QCTN23 =>
                interp_real             := CALC_INTERPOL(currentREAL0, currentREAL1);
                                                                -- Vertikale Interpolation wird
                     berechnet
                interp_img              := CALC_INTERPOL(currentIMG0,  currentIMG1);

                WR_ADDR                 := JUMP1CELL_BACK(WR_ADDR);
310             ADDR                    <= WR_ADDR;
                                                                                                    --
                     ===============================================
                DOUT                    <= WRITE_32_BITS_ON_RAM(interp_real, interp_img);
            when QCTN24 =>
                WR_ADDR                 :=INC_ADDR(WR_ADDR);    -- POSITIONIEREN
                WE                      <= LOWLEVEL;
                                                                                                    --
                     Zeiger auf SpeicherZELLE(ROW0)
315             ADDR                    <= RE_ADDR0;
                                                                                                    --
                     Zeiger auf SpeicherZELLE(ROW0)
                RE                      <= HIGHLEVEL;
            when QCTN25 =>                                          -- READ7
                roh_real0               := signed(DIN(R_MSB downto R_LSB));
                                                                        -- READ0, real0 aus der
                     ersten Zeile
                roh_img0                := signed(DIN(I_MSB downto I_LSB));
320             ADDR                    <= RE_ADDR1;
            when QCTN26 =>
                roh_real1               := signed(DIN(R_MSB downto R_LSB));
                                                                        -- READ0, real0 aus der
                     ersten Zeile
                roh_img1                := signed(DIN(I_MSB downto I_LSB));
                                                                        --
                     ===============================================
                interp_real             := CALC_INTERPOL(roh_real0, roh_real1);
                                                                        -- Vertikale Interpolation wird
                      berechnet
325             interp_img              := CALC_INTERPOL(roh_img0,  roh_img1);
                currentREAL0            := interp_real;
                                                                                                --
                     store Value for next interpol
                currentIMG0             := interp_img;

                RE                      <= LOWLEVEL;
                                                                                                    -- RE
                     LOW
330             RE_ADDR0                := JUMP2CELL(RE_ADDR0);
                                                                                    -- increment ADDR0,
                      jump to next value on row1
                RE_ADDR1                := JUMP2CELL(RE_ADDR1);
                                                                                    -- JUMP 2x READ1

                WR_ADDR                 := JUMP2CELL(WR_ADDR);
```

```
                    ADDR                   <= WR_ADDR;
335                 WE                     <= HIGHLEVEL;
                    DOUT                   <= WRITE_32_BITS_ON_RAM(interp_real, interp_img);
                when QCTN27 =>
                    interp_real            := CALC_INTERPOL(currentREAL0, currentREAL1);
                                                  -- Vertikale Interpolation wird berechnet
                    interp_img             := CALC_INTERPOL(currentIMG0,  currentIMG1);
340
                    WR_ADDR                :=JUMP1CELL_BACK(WR_ADDR);
                    ADDR                   <= WR_ADDR;
                                                                                          --
                        ================================================
                    DOUT                   <= WRITE_32_BITS_ON_RAM(interp_real, interp_img);
                when QCTN28 =>
345                 WR_ADDR                := INC_ADDR(WR_ADDR);    -- POSITIONIEREN
                    WE                     <= LOWLEVEL;
                                                                                          --
                        Zeiger auf SpeicherZELLE(ROW0)
                    ADDR                   <= RE_ADDR0;
                                                                                          --
                        Zeiger auf SpeicherZELLE(ROW0)
                    RE                     <= HIGHLEVEL;
                when QCTN29 =>       -- READ8
350                 roh_real0              := signed(DIN(R_MSB downto R_LSB));
                                              -- READ0, real0 aus der ersten Zeile
                    roh_img0               := signed(DIN(I_MSB downto I_LSB));
                    ADDR                   <= RE_ADDR1;
                when QCTN30 =>
                    roh_real1              := signed(DIN(R_MSB downto R_LSB));
                                              -- READ0, real0 aus der ersten Zeile
355                 roh_img1               := signed(DIN(I_MSB downto I_LSB));
                                                                                          --
                        ================================================
                    interp_real            := CALC_INTERPOL(roh_real0, roh_real1);
                                              -- Vertikale Interpolation wird berechnet
                    interp_img             := CALC_INTERPOL(roh_img0,  roh_img1);
                    currentREAL1           := interp_real;
                                                                  -- store Value for next
                        interpol
                    currentIMG1            := interp_img;
360
                    RE                     <= LOWLEVEL;
                                                                          -- RE LOW

                    WR_ADDR                := JUMP2CELL(WR_ADDR);
                    ADDR                   <= WR_ADDR;
365                 WE                     <= HIGHLEVEL;
                    DOUT                   <= WRITE_32_BITS_ON_RAM(interp_real, interp_img);
                when QCTN32 =>
                    interp_real            := CALC_INTERPOL(currentREAL0, currentREAL1);
                                              -- Vertikale Interpolation wird berechnet
                    interp_img             := CALC_INTERPOL(currentIMG0,  currentIMG1);
370
                    WR_ADDR                :=JUMP1CELL_BACK(WR_ADDR);
                    ADDR                   <= WR_ADDR;
                                                                                          --
                        ================================================
                    DOUT                   <= WRITE_32_BITS_ON_RAM(interp_real, interp_img);
                when QCTN33 =>
375                 if col_INT_DONE_INTERN = '0' then
                        WR_ADDR                := INC_ADDR(WR_ADDR);    -- POSITIONIEREN
                        WE                     <= LOWLEVEL;
                        RE                     <= LOWLEVEL;
                        --
380                     if (to_integer(unsigned(RE_ADDR1)) >= 288) then
                            MODL_READY             <= '1';
                            col_INT_DONE_INTERN <= '1';
                        else
                            RE_ADDR0               := JUM2NEXTROW_ADDR(RE_ADDR0);
385                         RE_ADDR1               := JUM2NEXTROW_ADDR(RE_ADDR1);
                            WR_ADDR                := JUM2NEXTROW_ADDR(WR_ADDR);
                        end if;
                    end if;
                when others => null;
390         end case;
        end if;
    end process;
end Behavioral;
```

Quellcode A.9: zweiD_DFT.vhd: Toplevel der 2D-DFT.

```
--------------------------------------------------------------------------------
-- Company: Diese Entity ist ein Teil-Komponent des 2D-DFT Modul
-- ist sowohl in 1D-DFT als auch invertiert i 2D-DFT eingesetzt
-- berechnet die 210 Speicherzelle des BRAMs.
```

```
5   -- Engineer:
    --
    -- Create Date: 10/04/2018 04:38:59 PM
    -- Design Name: Ada Koundoul Mastermikroelektronik
    -- Module Name: zweiD_DFT - Behavioral
10  -- Project Name: Masterarbeit ISAR
    -- Target Devices: Zedboard Xilinx
    -- Tool Versions:
    -- Description: zweiD_DFT Komponent
    --
15  -- Dependencies:
    --
    -- Revision: Hamburg 24.10.2018
    -- Revision 0.01 - File Created
    -- Additional Comments:
20  --
    ----------------------------------------------------------------------------------
    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    use ieee.NUMERIC_STD.all;
25  use work.FUNCTIONS_PKG.all;
    use work.DSP_PKG.all;

    package zweiD_DFT_PKG is
    component zweiD_DFT is
30    Port (
      CLK       : in std_logic;
      nRESET    : in std_logic;
      EN        : in bit;
      DIN       : in SLV;
35    modCtrl   : in modul;
      ADDR      : out addr_type;
      WE        : out std_logic;
      RE        : out std_logic;
      ready     : out std_logic;
40    DOUT      : out SLV
       );
    end component;
    end package;

45  library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    use ieee.NUMERIC_STD.all;
    use work.FUNCTIONS_PKG.all;
    use work.DSP_PKG.all;
50  use work.D_PKG.all;      -- D_MODOL
    use work.DD_PKG.all;     -- DD_MODOL

    -- Uncomment the following library declaration if using
    -- arithmetic functions with Signed or Unsigned values
55  --use IEEE.NUMERIC_STD.ALL;

    -- Uncomment the following library declaration if instantiating
    -- any Xilinx leaf cells in this code.
    --library UNISIM;
60  --use UNISIM.VComponents.all;

    entity zweiD_DFT is
      Port (
      CLK       : in std_logic;
65    nRESET    : in std_logic;
      EN        : in bit;
      DIN       : in SLV;
      modCtrl   : in modul;
      ADDR      : out addr_type;
70    WE        : out std_logic;
      RE        : out std_logic;
      ready     : out std_logic;
      DOUT      : out SLV
       );
75  end zweiD_DFT;

    architecture Behavioral of zweiD_DFT is
    signal rdy, sel                     : std_logic;        -- verbindet D_MOD & DD_MOD
    signal addr_D, addr_DD              : addr_type;
80  signal addr_dft                     : addr_type;
    signal DOUT_D, DOUT_DD              : SLV;
    signal DOUT_dft                     : SLV;
    signal WE_D, WE_DD, RE_D, RE_DD     : std_logic;
    signal WE_dft, RE_dft               : std_logic;
85  signal Cready                       : std_logic;
    signal hochohmig                    : std_logic;
    begin
    --
    sel       <= rdy;
90  hochohmig   <= 'Z';
    -- Verdrahten
    D_DFT   : D_MOD port map(
```

```
    CLK         => CLK,
    nRESET      => nRESET,
95  DIN         => DIN,
    modCtrl     => modCtrl,
    ADDR        => addr_D,
    WE          => WE_D,
    RE          => RE_D,
100 ready       => rdy,
    DOUT        => DOUT_D
    );
    --
    DD_DFT  : DD_MOD port map(
105 CLK         => CLK,
    nRESET      => nRESET,
    DIN         => DIN,
    start       => rdy,
    ADDR        => addr_DD,
110 WE          => WE_DD,
    RE          => RE_DD,
    ready       => Cready,
    DOUT        => DOUT_DD
    );
115 --==================================================================
    -- %%%%%%%%%%%%%%%%%%%%%%%%%%%% MUX %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    --==================================================================
    RE_dft      <= RE_D    when   sel = '0' else RE_DD;
    WE_dft      <= WE_D    when   sel = '0' else WE_DD;
120 addr_dft    <= addr_D  when   sel = '0' else addr_DD;
    DOUT_dft    <= DOUT_D  when   sel = '0' else DOUT_DD;
    --==================================================================
    -- %%%%%%%%%%%%%%%%%%%%%%%%%% OUTPUTS %%%%%%%%%%%%%%%%%%%%%%%%%%
    --==================================================================
125 RE          <= RE_dft   when modCtrl = dftMod and EN = '0' else 'Z';
    WE          <= WE_dft   when modCtrl = dftMod and EN = '0' else 'Z';
    DOUT        <= DOUT_dft when modCtrl = dftMod and EN = '0' else (others =>'Z');
    ADDR        <= addr_dft when modCtrl = dftMod and EN = '0' else (others =>'Z');
    ready       <= Cready   when modCtrl = dftMod else 'Z';
130 end Behavioral;
```

Quellcode A.10: D_MOD.vhd: Toplevel der 1D-DFT.

```
    ----------------------------------------------------------------------------------
    -- Company: Diese Entity ist ein Teil-Komponent des 2D-DFT Modul
    -- ist sowohl in 1D-DFT als auch invertiert i 2D-DFT eingesetzt
    -- berechnet die 210 Speicherzelle des BRAMs.
5   -- Engineer:
    --
    -- Create Date: 10/04/2018 04:38:59 PM
    -- Design Name: Ada Koundoul
    -- Module Name: operatUnit - Behavioral
10  -- Project Name: Masterarbeit ISAR
    -- Target Devices: Zedboard Xilinx
    -- Tool Versions:
    -- Description: 1D-DFT-Komponent
    --
15  -- Dependencies:
    --
    -- Revision: Hamburg 24.10.2018
    -- Revision 0.01 - File Created
    -- Additional Comments:
20  --
    ----------------------------------------------------------------------------------
    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    use ieee.NUMERIC_STD.all;
25  use work.FUNCTIONS_PKG.all;
    use work.DSP_PKG.all;

    package D_PKG is
    component D_MOD is
30  Port (
      CLK         : in std_logic;
      nRESET      : in std_logic;
      DIN         : in SLV;
      modCtrl     : in modul;
35    ADDR        : out addr_type;
      WE          : out std_logic;
      RE          : out std_logic;
      ready       : out std_logic;
      DOUT        : out SLV
40    );
    end component;
    end package;

    library IEEE;
```

```
45  use IEEE.STD_LOGIC_1164.ALL;
    use ieee.NUMERIC_STD.all;
    use work.FUNCTIONS_PKG.all;
    use work.DSP_PKG.all;
    use work.addUnit_PKG.all;
50  use work.operatUnit_PKG.all;

    -- Uncomment the following library declaration if using
    -- arithmetic functions with Signed or Unsigned values
    --use IEEE.NUMERIC_STD.ALL;
55
    -- Uncomment the following library declaration if instantiating
    -- any Xilinx leaf cells in this code.
    --library UNISIM;
    --use UNISIM.VComponents.all;
60
    entity D_MOD is
    Port (
      CLK       : in std_logic;
      nRESET    : in std_logic;
65    DIN       : in SLV;
      modCtrl   : in modul;
      ADDR      : out addr_type;
      WE        : out std_logic;
      RE        : out std_logic;
70    ready     : out std_logic;
      DOUT      : out SLV
        );
    end D_MOD;

75  architecture Behavioral of D_MOD is
    --
    signal rdy, ready_op, sel       : std_logic;
    signal RE_add, RE_op            : std_logic;
    signal WE_add, WE_op            : std_logic;
80  signal addr_add, addr_op        : addr_type;
    signal dout_add, dout_op        : SLV;
    --
    begin
    sel <= rdy;
85  -- Component Verdrahten
    ADD        : addUnit port map(
          CLK        => CLK,
          nRESET     => nRESET,
          DIN        => DIN,
90        modCtrl    => modCtrl,
          ADDR       => addr_add,
          WE         => WE_add,
          RE         => RE_add,
          ready      => rdy,
95        DOUT       => dout_add
    );
    --
    OPERATE      : operatUnit port map(
      CLK        => CLK,
100   nRESET     => nRESET,
      DIN        => DIN,
      start      => rdy,
      ADDR       => addr_op,
      WE         => WE_op,
105   RE         => RE_op,
      ready      => ready_op,
      DOUT       => dout_op
    );
    --===================================================================
110 -- %%%%%%%%%%%%%%%%%%%%%%%%%% MUX %%%%%%%%%%%%%%%%%%%%%%%%%%
    --===================================================================
    RE      <= RE_add    when sel = '0' else RE_op;
    WE      <= WE_add    when sel = '0' else WE_op;
    ADDR    <= addr_add  when sel = '0' else addr_op;
115 DOUT    <= dout_add  when sel = '0' else dout_op;
    ready   <= '0'       when sel = '0' else ready_op;
    end Behavioral;
```

Quellcode A.11: DD_MOD.vhd: Toplevel der 2D-DFT.

```
    ----------------------------------------------------------------------------------
    -- Company: Diese Entity ist ein Teil-Komponent des 2D-DFT Modul
    -- ist sowohl in 1D-DFT als auch invertiert i 2D-DFT eingesetzt
    -- berechnet die 210 Speicherzelle des BRAMs.
5   -- Engineer:
    --
    -- Create Date: 10/04/2018 04:38:59 PM
    -- Design Name: Ada Koundoul Mastermikroelektronik
    -- Module Name: 2D-DFT - Behavioral
```

```
10 -- Project Name: Masterarbeit ISAR
   -- Target Devices: Zedboard Xilinx
   -- Tool Versions:
   -- Description: 2D-DFT Komponent
   --
15 -- Dependencies:
   --
   -- Revision: Hamburg 24.10.2018
   -- Revision 0.01 - File Created
   -- Additional Comments:
20 --
   ----------------------------------------------------------------------------------
   library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
   use ieee.NUMERIC_STD.all;
25 use work.FUNCTIONS_PKG.all;
   use work.DSP_PKG.all;

   package DD_PKG is
   component DD_MOD is
30 Port (
     CLK       : in std_logic;
     nRESET    : in std_logic;
     DIN       : in SLV;
     start     : in std_logic;
35   ADDR      : out addr_type;
     WE        : out std_logic;
     RE        : out std_logic;
     ready     : out std_logic;
     DOUT      : out SLV
40     );
   end component;
   end package;

   library IEEE;
45 use IEEE.STD_LOGIC_1164.ALL;
   use ieee.NUMERIC_STD.all;
   use work.FUNCTIONS_PKG.all;
   use work.operatUnit2D_PKG.all;
   use work.addUnit2D_PKG.all;
50 use work.DSP_PKG.all;

   -- Uncomment the following library declaration if using
   -- arithmetic functions with Signed or Unsigned values
   --use IEEE.NUMERIC_STD.ALL;
55
   -- Uncomment the following library declaration if instantiating
   -- any Xilinx leaf cells in this code.
   --library UNISIM;
   --use UNISIM.VComponents.all;
60
   entity DD_MOD is
   Port (
     CLK       : in std_logic;
     nRESET    : in std_logic;
65   DIN       : in SLV;
     start     : in std_logic;
     ADDR      : out addr_type;
     WE        : out std_logic;
     RE        : out std_logic;
70   ready     : out std_logic;
     DOUT      : out SLV
       );
   end DD_MOD;

75 architecture Behavioral of DD_MOD is
   --
   signal rdy, ready_op, sel       : std_logic;
   signal RE_add, RE_op            : std_logic;
   signal WE_add, WE_op            : std_logic;
80 signal addr_add, addr_op        : addr_type;
   signal dout_add, dout_op        : SLV;
   --
   begin
   sel <= rdy;
85 -- Component Verdrahten
   ADD2D     : addUnit2D port map(
       CLK       => CLK,
       nRESET    => nRESET,
       DIN       => DIN,
90     start     => start,
       ADDR      => addr_add,
       WE        => WE_add,
       RE        => RE_add,
       ready     => rdy,
95     DOUT      => dout_add
   );
   --
```

```
      OPERATE2D    : operatUnit2D port map(
        CLK        => CLK,
100     nRESET     => nRESET,
        DIN        => DIN,
        start      => rdy,
        ADDR       => addr_op,
        WE         => WE_op,
105     RE         => RE_op,
        ready      => ready_op,
        DOUT       => dout_op
      );
      --=======================================================================
110   -- %%%%%%%%%%%%%%%%%%%%%%%% MUX-OUTPUT%%%%%%%%%%%%%%%%%%%%%%%%
      --=======================================================================
      RE      <= RE_add   when sel = '0' else RE_op;
      WE      <= WE_add   when sel = '0' else WE_op;
      ADDR    <= addr_add when sel = '0' else addr_op;
115   DOUT    <= dout_add when sel = '0' else dout_op;
      ready   <= '0'      when sel = '0' else ready_op;
      end Behavioral;
```

Quellcode A.12: addUnit.vhd: Berechnung der ersten Zeile der 1D-DFT. Das gleiche Modul wird für die 2D-DFT eingesetzt. Jedoch wird dafür ein anderer Speicherbereich angesprochen – Die Adressen ändern sich.

```
      ----------------------------------------------------------------------------------
      -- Company: Diese Entity ist ein Teil-Komponent des 2D-DFT Modul
      -- ist sowohl in 1D-DFT als auch invertiert i 2D-DFT eingesetzt
      -- berechnet die 210 Speicherzelle des BRAMs.
5     -- Engineer:
      --
      -- Create Date: 10/04/2018 04:38:59 PM
      -- Design Name: Ada Koundoul
      -- Module Name: operatUnit - Behavioral
10    -- Project Name: Masterarbeit ISAR
      -- Target Devices: Zedboard Xilinx
      -- Tool Versions:
      -- Description: addUnit.vhdl
      --
15    -- Dependencies:
      --
      -- Revision: Hamburg 24.10.2018
      -- Revision 0.01 - File Created
      -- Additional Comments:
20    --
      ----------------------------------------------------------------------------------
      library IEEE;
      use IEEE.STD_LOGIC_1164.ALL;
      use ieee.NUMERIC_STD.all;
25    use work.FUNCTIONS_PKG.all;
      use work.DSP_PKG.all;

      package addUnit_PKG is
      component addUnit is
30    Port (
        CLK        : in std_logic;
        nRESET     : in std_logic;
        DIN        : in SLV;
        modCtrl    : in modul;
35      ADDR       : out addr_type;
        WE         : out std_logic;
        RE         : out std_logic;
        ready      : out std_logic;
        DOUT       : out SLV
40      );
      end component;
      end package;

      library IEEE;
45    use IEEE.STD_LOGIC_1164.ALL;
      use ieee.NUMERIC_STD.all;
      use work.FUNCTIONS_PKG.all;
      use work.Operate_PKG.all;
      use work.DSP_PKG.all;
50

      -- Uncomment the following library declaration if using
      -- arithmetic functions with Signed or Unsigned values
      --use IEEE.NUMERIC_STD.ALL;
55
      -- Uncomment the following library declaration if instantiating
      -- any Xilinx leaf cells in this code.
      --library UNISIM;
```

```
    --use UNISIM.VComponents.all;
60
    entity addUnit is
    Port (
      CLK        : in std_logic;
      nRESET     : in std_logic;
65    DIN        : in SLV;
      modCtrl    : in modul;
      ADDR       : out addr_type;
      WE         : out std_logic;
      RE         : out std_logic;
70    ready      : out std_logic;
      DOUT       : out SLV
       );
    end addUnit;

75  architecture Behavioral of addUnit is
    signal QPLUS                                     : QCOUNTER;      -- 6 BITS UNSIGNED COUNTER
    signal Q                                         : QCOUNTER;
    signal cltr_op        :   std_logic              := '0';
    begin
80  --==================================================================
    --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SAVECOUNTER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    --==================================================================
    SchaltwerkCounter :   process(nRESET, CLK)
    begin
85      if nRESET = '0' or modCtrl /= dftMod then
            Q      <= QCTN0;
        elsif CLK'event and CLK = '1' then
            Q <= QPLUS;                              -- COUNT
        end if;
90  end process;
    --==================================================================
    --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COUNTER_P %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    --==================================================================
    SchaltnetzCounter : process(Q, nRESET, modCtrl)
95  variable QEND    : QCOUNTER   := QCTN33;            -- 31 Takte for a Round
    begin
        if nRESET = '0' or modCtrl /= dftMod then
            QPLUS      <= QCTN0;
        elsif Q = QEND then
100         if cltr_op = '0' then
                QPLUS    <= QCTN1;                          -- QPLUS ZURUECKSETZEN
            end if;
         else
            if cltr_op = '0' then
105             QPLUS    <= Q + 1;                          -- count RISING_EDGE(CLK)
            end if;
        end if;
    end process;
    --====================================================================
110 --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DFT_OPERATING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    --====================================================================
    DataProcessing : process(CLK, QPLUS)
    variable addr2Read          : addr_type      := "0001000000"; -- 64
    variable addr2Write         : addr_type      := "0100100001"; -- 289
115 variable realwert           : data_type      := (others => '0');
    variable imgwert           : data_type      := (others => '0');
    begin
        if CLK'event and CLK = '0' then
            case QPLUS is
120             when QCTN0 =>
                    -- Init Signale
                    RE        <= '0';
                    WE        <= '0';
                    ready     <= '0';
125                 cltr_op <= '0';
                    ADDR      <= (others => '0');
                    DOUT      <= (others => '0');
                    -- Init variable
                    addr2Read              := "0001000000"; -- 64
130                 addr2Write             := "0100100001"; -- 289
                    realwert               := (others => '0');
                    imgwert                := (others => '0');
                when QCTN1 =>
                    RE             <= '1';
135                 ADDR           <= addr2Read;         --0
                when QCTN2 =>
                    realwert       := data_type(DIN(rMSB downto rLSB));
                    imgwert        := data_type(DIN(iMSB downto iLSB));
                    addr2Read      := jump2nextAddr(addr2Read);
140             when QCTN3 =>
                    ADDR           <= addr2Read;         --1
                when QCTN4 =>
                    realwert       := add(realwert ,data_type(DIN(rMSB downto rLSB)));
                    imgwert        := add(imgwert, data_type(DIN(iMSB downto iLSB)));
145                 addr2Read      := jump2nextAddr(addr2Read);
                when QCTN5 =>
```

```
                    ADDR          <= addr2Read;       --2
                when QCTN6 =>
                    realwert      := add(realwert ,data_type(DIN(rMSB downto rLSB)));
150                 imgwert       := add(imgwert, data_type(DIN(iMSB downto iLSB)));
                    addr2Read     := jump2nextAddr(addr2Read);
                when QCTN7 =>
                    ADDR          <= addr2Read;       --3
                when QCTN8 =>
155                 realwert      := add(realwert ,data_type(DIN(rMSB downto rLSB)));
                    imgwert       := add(imgwert, data_type(DIN(iMSB downto iLSB)));
                    addr2Read     := jump2nextAddr(addr2Read);
                when QCTN9 =>
                    ADDR          <= addr2Read;       --4
160             when QCTN10 =>
                    realwert      := add(realwert ,data_type(DIN(rMSB downto rLSB)));
                    imgwert       := add(imgwert, data_type(DIN(iMSB downto iLSB)));
                    addr2Read     := jump2nextAddr(addr2Read);
                when QCTN11 =>
165                 ADDR          <= addr2Read;       --5
                when QCTN12 =>
                    realwert      := add(realwert ,data_type(DIN(rMSB downto rLSB)));
                    imgwert       := add(imgwert, data_type(DIN(iMSB downto iLSB)));
                    addr2Read     := jump2nextAddr(addr2Read);
170             when QCTN13 =>
                    ADDR          <= addr2Read;       --6
                when QCTN14 =>
                    realwert      := add(realwert ,data_type(DIN(rMSB downto rLSB)));
                    imgwert       := add(imgwert, data_type(DIN(iMSB downto iLSB)));
175                 addr2Read     := jump2nextAddr(addr2Read);
                when QCTN15 =>
                    ADDR          <= addr2Read;       --7
                when QCTN16 =>
                    realwert      := add(realwert ,data_type(DIN(rMSB downto rLSB)));
180                 imgwert       := add(imgwert, data_type(DIN(iMSB downto iLSB)));
                    addr2Read     := jump2nextAddr(addr2Read);
                when QCTN17 =>
                    ADDR          <= addr2Read;       --8
                when QCTN18 =>
185                 realwert      := add(realwert ,data_type(DIN(rMSB downto rLSB)));
                    imgwert       := add(imgwert, data_type(DIN(iMSB downto iLSB)));
                    addr2Read     := jump2nextAddr(addr2Read);
                when QCTN19 =>
                    ADDR          <= addr2Read;       --9
190             when QCTN20 =>
                    realwert      := add(realwert ,data_type(DIN(rMSB downto rLSB)));
                    imgwert       := add(imgwert, data_type(DIN(iMSB downto iLSB)));
                    addr2Read     := jump2nextAddr(addr2Read);
                when QCTN21 =>
195                 ADDR          <= addr2Read;       --10
                when QCTN22 =>
                    realwert      := add(realwert ,data_type(DIN(rMSB downto rLSB)));
                    imgwert       := add(imgwert, data_type(DIN(iMSB downto iLSB)));
                    addr2Read     := jump2nextAddr(addr2Read);
200             when QCTN23 =>
                    ADDR          <= addr2Read;       --11
                when QCTN24 =>
                    realwert      := add(realwert ,data_type(DIN(rMSB downto rLSB)));
                    imgwert       := add(imgwert, data_type(DIN(iMSB downto iLSB)));
205                 addr2Read     := jump2nextAddr(addr2Read);
                when QCTN25 =>
                    ADDR          <= addr2Read;       --12
                when QCTN26 =>
                    realwert      := add(realwert ,data_type(DIN(rMSB downto rLSB)));
210                 imgwert       := add(imgwert, data_type(DIN(iMSB downto iLSB)));
                    addr2Read     := jump2nextAddr(addr2Read);
                when QCTN27 =>
                    ADDR          <= addr2Read;       --13
                when QCTN28 =>
215                 realwert      := add(realwert ,data_type(DIN(rMSB downto rLSB)));
                    imgwert       := add(imgwert, data_type(DIN(iMSB downto iLSB)));
                    addr2Read     := jump2nextAddr(addr2Read);
                when QCTN29 =>
                    ADDR         <= addr2Read;       --14
220             when QCTN30 =>
                    realwert      := add(realwert ,data_type(DIN(rMSB downto rLSB)));
                    imgwert       := add(imgwert, data_type(DIN(iMSB downto iLSB)));
                when QCTN31 =>
                    RE            <= '0';
225             when QCTN32 =>
                    WE            <= '1';
                    ADDR          <= addr2Write;
                    DOUT          <= writeDATA(realwert, imgwert);
                when QCTN33 =>
230                 if cltr_op = '0' then
                        WE            <= '0';
                        realwert     := (others => '0');
                        imgwert      := (others => '0');
                        addr2Write   := addr_type(unsigned(addr2Write)+15);
```

```
235                         if to_integer(unsigned(addr2Read)) >= 288  then
                                ready <= '1';
                                cltr_op  <= '1';
                            else
                                addr2Read   := addr_type(unsigned(addr2Read)-209);
240                         end if;
                        else
                            ready <= '1';
                            --realwert    := (others => '0');                 -- Latch vermeiden
                            --imgwert     := (others => '0');
245                     end if;
                    when others => null;
                end case;
            end if;
        end process;
250     end Behavioral;
```

Quellcode A.13: operatUnit.vhd: Berechnung der restlichen Einträge der 1D-DFT, nachdem die erste Zeile berechnet wurde. Das gleiche Modul wird für die 2D-DFT eingesetzt. Jedoch wird dafür ein anderer Speicherbereich angesprochen – Die Adressen ändern sich.

```
    ----------------------------------------------------------------------------------
    -- Company: Diese Entity ist ein Teil-Komponent des 2D-DFT Modul
    -- ist sowohl in 1D-DFT als auch invertiert i 2D-DFT eingesetzt
    -- berechnet die 210 Speicherzelle des BRAMs.
5   -- Engineer:
    --
    -- Create Date: 10/04/2018 04:38:59 PM
    -- Design Name: Ada Koundoul
    -- Module Name: operatUnit - Behavioral
10  -- Project Name: Masterarbeit ISAR
    -- Target Devices: Zedboard Xilinx
    -- Tool Versions:
    -- Description:
    --
15  -- Dependencies:
    --
    -- Revision: Hamburg 24.10.2018
    -- Revision 0.01 - File Created
    -- Additional Comments:
20  --
    ----------------------------------------------------------------------------------
    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    use ieee.NUMERIC_STD.all;
25  use work.FUNCTIONS_PKG.all;
    use work.DSP_PKG.all;

    package operatUnit_PKG is
    component operatUnit is
30  Port (
      CLK       : in std_logic;
      nRESET    : in std_logic;
      DIN       : in SLV;
      start     : in std_logic;
35    ADDR      : out addr_type;
      WE        : out std_logic;
      RE        : out std_logic;
      ready     : out std_logic;
      DOUT      : out SLV
40    );
    end component;
    end package;

    library IEEE;
45  use IEEE.STD_LOGIC_1164.ALL;
    use ieee.NUMERIC_STD.all;
    use work.FUNCTIONS_PKG.all;
    use work.Operate_PKG.all;
    use work.DSP_PKG.all;
50
    -- Uncomment the following library declaration if using
    -- arithmetic functions with Signed or Unsigned values
    --use IEEE.NUMERIC_STD.ALL;

55  -- Uncomment the following library declaration if instantiating
    -- any Xilinx leaf cells in this code.
    --library UNISIM;
    --use UNISIM.VComponents.all;

60  entity operatUnit is
```

```
    Port (
      CLK        : in std_logic;
      nRESET     : in std_logic;
      DIN        : in SLV;
65    start      : in std_logic;
      ADDR       : out addr_type;
      WE         : out std_logic;
      RE         : out std_logic;
      ready      : out std_logic;
70    DOUT       : out SLV
      );
    end operatUnit;

    architecture Behavioral of operatUnit is
75  signal QPLUS                       : QCOUNTER;    -- 6 BITS UNSIGNED COUNTER
    signal Q                           : QCOUNTER;
    signal cltr_op     : std_logic     := '0';
    begin
    --===========================================================================
80  --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SAVECOUNTER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    --===========================================================================
    SchaltwerkCounter :  process(nRESET, CLK)
    begin
        if nRESET = '0' or start /= '1' then
85          Q      <= QCTN0;
        elsif CLK'event and CLK = '1' then
            Q <= QPLUS;                     -- COUNT
        end if;
    end process;
90  --===========================================================================
    --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COUNTER_P %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    --===========================================================================
    SchaltnetzCounter : process(Q, nRESET, start)
    variable QEND    : QCOUNTER  := QCTN49;        -- 31 Takte for a Round
95  begin
        if nRESET = '0' or start /= '1' then
            QPLUS      <= QCTN0;
        elsif Q = QEND then
            if cltr_op = '0' then
100             QPLUS    <= QCTN1;                  -- QPLUS ZURUECKSETZEN
            end if;
         else
           if cltr_op = '0' then
                QPLUS    <= Q + 1;                  -- count RISING_EDGE(CLK)
105        end if;
        end if;
    end process;
    --=============================================================================
    --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DFT_OPERATING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
110 --=============================================================================
    DataProcessing : process(CLK, QPLUS)
    variable addr2Write          : addr_type          := "0100100010";     -- write ab addr 290
    variable startAddrWrite      : addr_type          := "0100100010";     -- write ab addr 290
    variable addrUp2Down         : addr_type          := "0001001111";     -- read ab addr 79
115 variable addrDown2Up         : addr_type          := "0100010010";     -- read ab addr 274
    variable addr1Row            : addr_type          := "0001000000";     --  64 addr der Zellen
          der ersten Interpolationszeile
    variable addr4TW             : addr_type          := "1011100011";     -- 739
    --
    variable realwert            : data_type          := (others => '0');  -- speichert vorl\"
         aufig den gelesenen real-Wert
120 variable imgwert             : data_type          := (others => '0');  -- speichert vorl\"
         aufig den gelesenen img-Wert
    variable realwert_im         : data_type          := (others => '0');
    variable imgwert_re          : data_type          := (others => '0');
    variable currentreal         : data_type          := (others => '0');
    variable currentimg          : data_type          := (others => '0');
125 variable cellCounter         : unsigned(3 downto 0)  := (others =>'0');    -- count anzahl Cell
          in Row
    variable wholeRESULT         : unsigned(11 downto 0)  := (others =>'0');    -- counter fuer
         alle geschriebenen Zellen
    variable currentRound        : std_logic          := '0';
    begin
        if CLK'event and CLK = '0' then
130         case QPLUS is
                when QCTN0 =>                       -- if nRESET  ==> Z\"ahlZustand QCTN0
                  -- Init signale
                    RE              <= '0';
                    WE              <= '0';
135                 ready           <= '0';
                    cltr_op         <= '0';
                    ADDR            <= (others => '0');
                    DOUT            <= (others => '0');
                  -- Init variable
140                 addr2Write      := "0100100010";      -- write ab addr 290
                    startAddrWrite  := "0100100010";      -- write ab addr 290
                    addrUp2Down     := "0001001111";      -- read ab addr 79
                    addrDown2Up     := "0100010010";      -- read ab addr 274
```

```
        addr1Row          := "0001000000";      -- 64+
145     addr4TW           := "1011100011";      -- 739
        wholeRESULT       := (others => '0');   -- counter for whole result; termination
            condition
        realwert          := (others => '0');   -- holds the real values in s1q10 format
        imgwert           := (others => '0');   -- holds the imag values in s1q10 format
        realwert_im       := (others => '0');   -- holds the real values in s1q10 format
150     imgwert_re        := (others => '0');   -- holds the imag values in s1q10 format
        currentreal       := (others => '0');
        currentimg        := (others => '0');
        cellCounter       := (others => '0');
        currentRound      := '0';
155
    --========================================================================
    --  Set Address to get the first values of interpolated
    --  data. This address holds the values of first row in matrix
    --========================================================================
160 when QCTN1 =>                                          -- cellMult0
        RE                <= '1';
        ADDR              <= addrUp2Down;       -- 79
    --========================================================================
    --  Values of first row of interpolated data
165 --  are multiplied by 1. => Just copy values to
    --                         realwert
    --                         imagwert
    --  and jump to the next row (Address+15)
    --========================================================================
170 when QCTN2 =>
        realwert          := data_type(DIN(rMSB downto rLSB));
        imgwert           := data_type(DIN(iMSB downto iLSB));
        realwert_im       := data_type(DIN(rMSB downto rLSB));
        imgwert_re        := data_type(DIN(iMSB downto iLSB));
175     addrUp2Down       := jump2nextAddr(addrUp2Down);      -- 94
    when QCTN3 =>
        ADDR              <= addrDown2Up;       -- 274 Interpolationswert Adresse der Beginn
             letzte Zeile
    when QCTN4 =>
        if currentRound = '0' then
180         imgwert           := add(imgwert, invert(data_type(DIN(iMSB downto iLSB))));
                     -- fuer das Produkt mit real Twiddle
            realwert_im       := add(realwert_im, invert(data_type(DIN(rMSB downto rLSB))
                ));
        else
            imgwert           := add(data_type(DIN(iMSB downto iLSB)),invert(imgwert));
            realwert_im       := add(data_type(DIN(rMSB downto rLSB)),invert(realwert_im)
                );
185     end if;
        imgwert_re        := add(imgwert_re ,data_type(DIN(iMSB downto iLSB)));     -- a
        realwert          := add(realwert ,data_type(DIN(rMSB downto rLSB)));     -- a
        addrDown2Up       := jumpBack2nextAddr(addrDown2Up);      -- 259              -- ende
            Interpolationswert Adresse der Beginn vorletzte Zeile
                                                                                      --
                                                                                      64+15*13
190 when QCTN5 =>          -- read Twiddle Faktor
        ADDR              <= addr4TW;       -- 739
    when QCTN6 =>     -- Product bilden
        currentreal       := add(          mult(realwert ,data_type(DIN(rMSB downto rLSB))),
                             invert(mult(imgwert, data_type(DIN(iMSB downto iLSB)))))
                                 ;                                      -- ax-by
195     currentimg        := add( mult(realwert_im, data_type(DIN(iMSB downto iLSB))),
                             mult(imgwert_re ,data_type(DIN(rMSB downto rLSB)))); --
                                 i(ay+bx)
    when QCTN7 =>
        addr4TW           := addr_type(unsigned(addr4TW)+1);                    -- incremment
             740
        realwert          :=(others => '0');
200     imgwert           :=(others => '0');
        ADDR              <= addrUp2Down;                            --94
    when QCTN8 =>                                               -- cellMult1
        realwert          := data_type(DIN(rMSB downto rLSB));
        imgwert           := data_type(DIN(iMSB downto iLSB));
205     realwert_im       := data_type(DIN(rMSB downto rLSB));
        imgwert_re        := data_type(DIN(iMSB downto iLSB));
        addrUp2Down       := jump2nextAddr(addrUp2Down);      -- 109
    when QCTN9 =>
        ADDR              <= addrDown2Up;                        -- 259
210 when QCTN10 =>
        if currentRound = '0' then
            imgwert           := add(imgwert, invert(data_type(DIN(iMSB downto iLSB))));
            realwert_im       := add(realwert_im, invert(data_type(DIN(rMSB downto rLSB))
                ));
        else
215         imgwert           := add(data_type(DIN(iMSB downto iLSB)),invert(imgwert));
            realwert_im       := add(data_type(DIN(rMSB downto rLSB)),invert(realwert_im)
                );
        end if;
        imgwert_re        := add(imgwert_re ,data_type(DIN(iMSB downto iLSB)));     -- a
```

```
                realwert        := add(realwert ,data_type(DIN(rMSB downto rLSB)));    -- a
220             addrDown2Up     := jumpBack2nextAddr(addrDown2Up);    -- 259             -- ende
                    Interpolationswert Adresse der Beginn vorletzte Zeile
            when QCTN11 =>       -- read Twiddel Faktor
                ADDR            <= addr4TW;     -- 740
            when QCTN12 =>
                currentreal     := add(add(         mult(realwert ,data_type(DIN(rMSB downto rLSB
                    ))),
225                                      invert(mult(imgwert, data_type(DIN(iMSB downto iLSB)))))
                                             ,
                                         currentreal);                      -- ax-by
                currentimg      := add(add( mult(realwert_im, data_type(DIN(iMSB downto iLSB))),
                                         mult(imgwert_re ,data_type(DIN(rMSB downto rLSB)))),
                                         currentimg); --  i(ay+bx)
230         when QCTN13 =>
                addr4TW         := addr_type(unsigned(addr4TW)+1);                    --
                    incremment 741
                ADDR            <= addrUp2Down;                    --109
            when QCTN14 =>                                         -- cellMult2
                realwert        := data_type(DIN(rMSB downto rLSB));
235             imgwert         := data_type(DIN(iMSB downto iLSB));
                realwert_im     := data_type(DIN(rMSB downto rLSB));
                imgwert_re      := data_type(DIN(iMSB downto iLSB));
                addrUp2Down     := jump2nextAddr(addrUp2Down);     -- 124
            when QCTN15 =>
240             ADDR            <= addrDown2Up;                        --244
            when QCTN16 =>
                if currentRound = '0' then
                    imgwert         := add(imgwert, invert(data_type(DIN(iMSB downto iLSB))));
                    realwert_im     := add(realwert_im, invert(data_type(DIN(rMSB downto rLSB))
                        ));
245             else
                    imgwert         := add(data_type(DIN(iMSB downto iLSB)),invert(imgwert));
                    realwert_im     := add(data_type(DIN(rMSB downto rLSB)),invert(realwert_im)
                        );
                end if;
                imgwert_re      := add(imgwert_re ,data_type(DIN(iMSB downto iLSB)));    -- a
250             realwert        := add(realwert ,data_type(DIN(rMSB downto rLSB)));    -- a
                addrDown2Up     := jumpBack2nextAddr(addrDown2Up);    -- 259             -- ende
                    Interpolationswert Adresse der Beginn vorletzte Zeile
            when QCTN17 =>
                ADDR            <= addr4TW;     -- 741
            when QCTN18 =>
255             currentreal     := add(add(         mult(realwert ,data_type(DIN(rMSB downto rLSB
                    ))),
                                             invert(mult(imgwert, data_type(DIN(iMSB downto iLSB)
                                                 )))),
                                                                                              currentreal
                                                                                                  );
                currentimg      := add(add(          mult(realwert_im, data_type(DIN(iMSB downto
                    iLSB))),
                                                     mult(imgwert_re ,data_type(DIN(rMSB downto
                                                         rLSB)))),
260                                                  currentimg); --  i(ay+bx)
            when QCTN19 =>
                addr4TW         := addr_type(unsigned(addr4TW)+1);                    --
                    incremment 742
                ADDR            <= addrUp2Down;                    --124
            when QCTN20 =>                                         -- cellMult3
265             realwert        := data_type(DIN(rMSB downto rLSB));
                imgwert         := data_type(DIN(iMSB downto iLSB));
                realwert_im     := data_type(DIN(rMSB downto rLSB));
                imgwert_re      := data_type(DIN(iMSB downto iLSB));
                addrUp2Down     := jump2nextAddr(addrUp2Down);     -- 139
270         when QCTN21 =>
                ADDR            <= addrDown2Up;                        --229
            when QCTN22 =>
                if currentRound = '0' then
                    imgwert         := add(imgwert, invert(data_type(DIN(iMSB downto iLSB))));
275                 realwert_im     := add(realwert_im, invert(data_type(DIN(rMSB downto rLSB))
                        ));
                else
                    imgwert         := add(data_type(DIN(iMSB downto iLSB)),invert(imgwert));
                    realwert_im     := add(data_type(DIN(rMSB downto rLSB)),invert(realwert_im)
                        );
                end if;
280             imgwert_re      := add(imgwert_re ,data_type(DIN(iMSB downto iLSB)));    -- a
                realwert        := add(realwert ,data_type(DIN(rMSB downto rLSB)));    -- a
                addrDown2Up     := jumpBack2nextAddr(addrDown2Up);    -- 259             -- ende
                    Interpolationswert Adresse der Beginn vorletzte Zeile
            when QCTN23 =>
                ADDR            <= addr4TW;     -- 742
285         when QCTN24 =>
                currentreal     := add(add(         mult(realwert ,data_type(DIN(rMSB downto rLSB
                    ))),
                                         invert(mult(imgwert, data_type(DIN(iMSB downto iLSB)))))
                                             ,
                                         currentreal);                      -- ax-by
```

```
                        currentimg       := add(add( mult(realwert_im , data_type(DIN(iMSB downto iLSB))),
290                                             mult(imgwert_re ,data_type(DIN(rMSB downto rLSB)))),
                                                currentimg); --  i(ay+bx)
                  when QCTN25 =>
                        addr4TW          := addr_type(unsigned(addr4TW)+1);                  --
                            incremment 743
                        ADDR             <= addrUp2Down;                     --139
295               when QCTN26 =>                                                 -- cellMult4
                        realwert         := data_type(DIN(rMSB downto rLSB));
                        imgwert          := data_type(DIN(iMSB downto iLSB));
                        realwert_im      := data_type(DIN(rMSB downto rLSB));
                        imgwert_re       := data_type(DIN(iMSB downto iLSB));
300                     addrUp2Down      := jump2nextAddr(addrUp2Down);      -- 154
                  when QCTN27 =>
                        ADDR             <= addrDown2Up;                         --214
                  when QCTN28 =>
                        if currentRound = '0' then
305                         imgwert          := add(imgwert, invert(data_type(DIN(iMSB downto iLSB))));
                            realwert_im      := add(realwert_im, invert(data_type(DIN(rMSB downto rLSB))
                                ));
                        else
                            imgwert          := add(data_type(DIN(iMSB downto iLSB)),invert(imgwert));
                            realwert_im      := add(data_type(DIN(rMSB downto rLSB)),invert(realwert_im)
                                );
310                     end if;
                        imgwert_re       := add(imgwert_re ,data_type(DIN(iMSB downto iLSB)));     -- a
                        realwert         := add(realwert ,data_type(DIN(rMSB downto rLSB)));     -- a
                        addrDown2Up      := jumpBack2nextAddr(addrDown2Up);     -- 259
                  when QCTN29 =>
315                     ADDR             <= addr4TW;           -- 743
                  when QCTN30 =>
                        currentreal      := add(add(         mult(realwert ,data_type(DIN(rMSB downto rLSB
                            ))),
                                                invert(mult(imgwert, data_type(DIN(iMSB downto iLSB)))))
                                                    ,
                                                currentreal);
320                     currentimg       := add(add( mult(realwert_im , data_type(DIN(iMSB downto iLSB))),
                                                mult(imgwert_re ,data_type(DIN(rMSB downto rLSB)))),
                                                currentimg); --  i(ay+bx)
                  when QCTN31 =>
                        addr4TW          := addr_type(unsigned(addr4TW)+1);                  --
                            incremment 744
325                     ADDR        <= addrUp2Down;                     --154
                  when QCTN32 =>                                                 -- cellMult5
                        realwert      := data_type(DIN(rMSB downto rLSB));
                        imgwert       := data_type(DIN(iMSB downto iLSB));
                        realwert_im   := data_type(DIN(rMSB downto rLSB));
330                     imgwert_re    := data_type(DIN(iMSB downto iLSB));
                        addrUp2Down   := jump2nextAddr(addrUp2Down);      -- 169
                  when QCTN33 =>
                        ADDR        <= addrDown2Up;                         --199
                  when QCTN34 =>
335                     if currentRound = '0' then
                            imgwert          := add(imgwert, invert(data_type(DIN(iMSB downto iLSB))));
                            realwert_im      := add(realwert_im, invert(data_type(DIN(rMSB downto rLSB))
                                ));
                        else
                            imgwert          := add(data_type(DIN(iMSB downto iLSB)),invert(imgwert));
340                         realwert_im      := add(data_type(DIN(rMSB downto rLSB)),invert(realwert_im)
                                );
                        end if;
                        imgwert_re       := add(imgwert_re ,data_type(DIN(iMSB downto iLSB)));     -- a
                        realwert         := add(realwert ,data_type(DIN(rMSB downto rLSB)));     -- a
                        addrDown2Up      := jumpBack2nextAddr(addrDown2Up);     -- 259          -- ende
                            Interpolationswert Adresse der Beginn vorletzte Zeile
345               when QCTN35 =>
                        ADDR        <= addr4TW;           -- 744
                  when QCTN36 =>
                        currentreal      := add(add(         mult(realwert ,data_type(DIN(rMSB downto rLSB
                            ))),
                                                invert(mult(imgwert, data_type(DIN(iMSB downto iLSB)))))
                                                    ,
350                                             currentreal);                        -- ax-by
                        currentimg       := add(add( mult(realwert_im , data_type(DIN(iMSB downto iLSB))),
                                                mult(imgwert_re ,data_type(DIN(rMSB downto rLSB)))),
                                                currentimg); --  i(ay+bx)
                  when QCTN37 =>
355                     addr4TW          := addr_type(unsigned(addr4TW)+1);                  --
                            incremment 745
                        ADDR             <= addrUp2Down;                     --169
                  when QCTN38 =>                                                 -- cellMult6
                        realwert      := data_type(DIN(rMSB downto rLSB));
                        imgwert       := data_type(DIN(iMSB downto iLSB));
360                     realwert_im   := data_type(DIN(rMSB downto rLSB));
                        imgwert_re    := data_type(DIN(iMSB downto iLSB));
                  when QCTN39 =>
                        ADDR        <= addrDown2Up;                         --184
                  when QCTN40 =>
```

```
365                 if currentRound = '0' then
                        imgwert          := add(imgwert, invert(data_type(DIN(iMSB downto iLSB))));
                        realwert_im      := add(realwert_im, invert(data_type(DIN(rMSB downto rLSB))
                            ));
                    else
                        imgwert          := add(data_type(DIN(iMSB downto iLSB)),invert(imgwert));
370                     realwert_im      := add(data_type(DIN(rMSB downto rLSB)),invert(realwert_im)
                            );
                    end if;
                    imgwert_re       := add(imgwert_re ,data_type(DIN(iMSB downto iLSB)));     -- a
                    realwert         := add(realwert ,data_type(DIN(rMSB downto rLSB)));     -- a
                when QCTN41 =>
375                 ADDR          <= addr4TW;           -- 745
                when QCTN42 =>
                    currentreal      := add(add(         mult(realwert ,data_type(DIN(rMSB downto rLSB
                        ))),
                                              invert(mult(imgwert, data_type(DIN(iMSB downto iLSB))))
                                              ,
                                              currentreal);                     -- ax-by
380                 currentimg       := add(add( mult(realwert_im, data_type(DIN(iMSB downto iLSB))),
                                              mult(imgwert_re ,data_type(DIN(rMSB downto rLSB)))),
                                              currentimg); --   i(ay+bx)
                when QCTN43 =>
                    ADDR          <= addr1Row;          -- 64 from first interpolationrow
385             when QCTN44 =>
                    currentreal      := add(currentreal ,data_type(DIN(rMSB downto rLSB)));     -- a
                    currentimg       := add(currentimg, data_type(DIN(iMSB downto iLSB)));     -- ib
                when QCTN45 =>
                    RE          <= '0';
390                 wholeRESULT     := wholeRESULT + 1;                -- Zelle increment
                    cellCounter     := cellCounter + 1;                -- Zelle increment
                when QCTN46 =>
                    WE            <= '1';
                    ADDR          <= addr2Write;
395                 DOUT          <= writeDATA(currentreal,currentimg);
                when QCTN47 =>
                    WE            <= '0';
                    currentreal  :=(others => '0');
                    currentimg   :=(others => '0');
400                 if (wholeRESULT >= "000011010010") then                             --
                        wholeRESULT = 210 gebildete Zellen
                        cltr_op       <= '1';                                      -- ende der
                            Gesamtverarbeitung ... Takt prozess
                        ready         <= '1';
                    end if;
                when QCTN48 =>
405                 realwert         :=(others => '0');
                    imgwert          :=(others => '0');
                    addrUp2Down     := addr_type(unsigned(addrUp2Down)-89);           -- neuer
                        Spaltenanfang
                    addrDown2Up     := addr_type(unsigned(addrDown2Up)+91);           -- neues
                        SpaltenEnde
                    addr1Row        := addr_type(unsigned(addr1Row)+1);               -- neue
                        Spalte
410                 addr2Write      := addr_type(unsigned(addr2Write)+15);             -- next
                        addrCell 2 write
                    addr4TW         := addr_type(unsigned(addr4TW)-6);                -- TWaddr
                        reset
                when QCTN49 =>
                    if (wholeRESULT >= "000001101001") then
                        currentRound   := '1';
415                 end if;
                    if (cellCounter >= "1111") then
                        addr2Write     :=(others => '0');
                        startAddrWrite:= addr_type(unsigned(startAddrWrite)+1);        -- next
                            addrCell 2 write
                        addr2Write     := startAddrWrite;                             -- next
                            addrCell 2 write
420
                        cellCounter    := (others => '0');
                        addrUp2Down    := "0001001111";                               -- reset
                            read ab addr 79
                        addrDown2Up    := "0100010010";                               -- reset
                            read ab addr 274
                        addr1Row       := "0001000000";                               -- reset
                            read ab addr 64
425                     if  (wholeRESULT    < "000001101001") then
                            addr4TW        := addr_type(unsigned(addr4TW)+7);
                        elsif (wholeRESULT >= "000001111000") then
                            addr4TW        := addr_type(unsigned(addr4TW)-7);
                        end if;
430                 end if;
                when others => null;
            end case;
        end if;
    end process;
435 end Behavioral;
```

Selbstständigkeitserklärung

Hiermit versichere ich, Ada Koundoul, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(6) PO_Ma_MeS ohne fremde Hilfe selbständig 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, 24. Oktober 2018