

CHIPIMPLEMENTATION EINER
ZWEIDIMENSIONALEN
FOURIERTRANSFORMATION FÜR DIE
AUSWERTUNG EINES SENSOR-ARRAYS

THOMAS LATTMANN

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
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. Jürgen Vollmer

Abgegeben am 23.04.2018

Thomas Lattmann

Thema der Bachelorarbeit

Chipimplementation einer zweidimensionalen Fouriertransformation für die Auswertung eines Sensor-Arrays

Stichworte

Cadence, ASIC, zweidimensionale diskrete Fouriertransformation (2D-DFT), Sensor-Array

Kurzzusammenfassung

In der Bachelorarbeit wird eine 8x8 zweidimensionale Fourier Transformation in VHDL für den Einsatz als Teilmodul auf einem ASIC entwickelt. Dabei wird das Chipdesign-Tool Cadence verwendet. Die Transformation wird durch eine Matrixmultiplikation realisiert und hinsichtlich Taktzyklen und Flächenbedarf optimiert, sodass ein minimaler Aufwand erforderlich ist. Ferner werden Tests zur Funktionalität durchgeführt und der Floorplan erstellt.

Thomas Lattmann

Title of the bachelor thesis

Chip implementation of a two dimensional Fourier transform for the analysis of a sensor array

Keywords

Cadence, ASIC, two dimensional discrete Fourier transform (2D-DFT), sensor array

Abstract

In this bachelor thesis a 8x8 twodimensional Fourier transform is developed in VHDL as a partial modul for the usage on an ASIC. For the implementation the chipdesign tool Cadence is used. The transform is realized as a matrix multiplication and optimized in terms of clock cycles and required area. Functionality test are executed and the floorplan is created.

Inhaltsverzeichnis

1. Einleitung	V
1.1. Motivation	V
1.2. Stand der Technik	V
1.3. Ziel dieser Arbeit	1
2. Grundlagen	2
2.1. Binäre Zahlendarstellung von Festkommazahlen	2
2.1.1. Integer-Zahl im 1er-Komplement	2
2.1.2. Integer-Zahl im 2er-Komplement	3
2.1.3. Darstellung dualer Zahlen im SQ-Format	3
2.1.4. Numerische Ungenauigkeiten	4
2.2. Mathematische Grundlagen	4
2.2.1. Komplexe Multiplikation	4
2.2.2. Matrixmultiplikation	5
2.3. Fourierreihenentwicklung	5
2.4. Fouriertransformation	7
2.4.1. Diskrete Fouriertransformation (DFT)	7
2.4.2. Summen- und Matrizenschreibweise der DFT	8
2.4.3. 2D-DFT mit reellen Eingangswerten	9
2.4.4. Berechnung der schnellen Fouriertransformation	10
2.4.5. Inverse DFT	12
2.5. Diskrete Kosinus Transformation (DCT)	12
3. Analyse	14
3.1. Bewertung verschiedener DFT- und DCT-Größen	14
3.1.1. Bewertung verschiedener DCT-Größen	15
3.1.2. Bewertung verschiedener DFT-Größen	15
3.1.3. Bewertungsfazit	15
3.2. Betrachtung der 8x8-DFT	16
3.3. Einordnung des Rechenaufwands	19
3.3.1. 8x8-DFT mit komplexen Eingangswerten	19
3.3.2. 8x8-DFT mit reellen Eingangswerten	20
3.3.3. Direkte Multiplikation zweier 8x8 Matrizen mit komplexen Werten	20
3.3.4. Betrachtung des Butterfly-Algorithmus für 8 Eingangswerte	21
3.3.5. Fazit der Berechnungs-Gegenüberstellungen	21

4. Entwurf	23
4.1. Projekt- und Programmstruktur	23
4.1.1. Vorüberlegungen aus Hardwaresicht	23
4.1.2. VHDL-Bibliotheken	24
4.1.3. Vorüberlegungen zum Programmablauf	24
4.1.4. Struktureller Aufbau	24
4.2. Entwicklung der 2D-DFT-Komponente	25
4.2.1. Optimierte 8x8 DFT als Matrixmultiplikation	25
4.2.2. Berechnungsschema und benötigte Takte der Ergebnisse	26
4.2.3. Programmablauf der 1D-DFT	28
4.2.4. Entwicklung von der 1D-DFT zur 2D-DFT	29
4.2.5. Zusammenhang von DFT und IDFT bei der Matrixmultiplikation	30
4.3. Syntheseergebnisse von Teilkomponenten	31
4.3.1. 13 Bit Konstantenmultiplizierer	31
4.3.2. Bildung des 2er-Komplements eines 13 Bit Vektors	31
4.3.3. 13 Bit Addierer	32
4.3.4. Vergleich der Syntheseergebnisse	32
4.3.5. Gegenüberstellung der Konstantenmultiplikation und der Bildung des 2er-Komplements	34
4.4. Schema der Zustandsfolge	34
4.5. UML-Diagramm	36
5. Evaluation	39
5.1. Simulation der 2D-DFT	39
5.2. Test des VHDL-Implementation der 2D-DFT	41
5.2.1. Matrixmultiplikation mit ganzzahligen Faktoren, 1D-DFT und 2D-DFT	41
5.2.2. Automatisierter Testdurchlauf	41
5.2.3. Test der 2D-DFT mit realen Eingangswerten	44
5.3. Zeitabschätzung als Winkelsensor im Antriebsmotor eines Elektroautos	45
5.4. Chipdesign	45
5.4.1. Syntheseergebnis der 2D-DFT-Einheit	46
5.4.2. Floorplan der 2D-DFT-Einheit mit 3 Metalllagen und 350nm Struktur- größe	48
6. Schlussfolgerungen	51
6.1. Zusammenfassung	51
6.2. Bewertung und Fazit	51
6.3. Ausblick	52
Abkürzungsverzeichnis	54
Abbildungsverzeichnis	56
Tabellenverzeichnis	57

Literatur	58
Anhang	60
A. Matlab-Skripte	60
A.1. Skript zur Bewertung von Twiddlefaktormatrizen	60
A.2. Twiddlefaktormatrix im S1Q10-Format	64
B. Gate-Reports der Syntheseergebnisse	69
B.1. Gate-Report des 13 Bit Konstantenmultiplizierers	69
B.2. Gate-Report des 13 Bit Multiplizierers	70
B.3. Gate-Report des 12 Bit Addierers	70
B.4. Gate-Report des 13 Bit Negierers	71
B.5. Gate-Report der 2D-DFT	72
C. VHDL-Code	74
C.1. Konstantendeklarationen	74
C.2. Datentypendeklarationen	74
C.3. Testwerte aus Datei einlesen	75
C.4. Ergebnisse in Datei schreiben	79
C.5. Berechnung der 2D-DFT	83
D. Testumgebung	98
D.1. Bash-Skript zum Ausführen des Tests	98
D.2. Skript zum Kompilieren und Simulieren des VHDL-Programms	98
D.3. Matlab-Skript zur Berechnung der Referenzwerte und Vergleich der Ergebnisse	100
E. Cadence Tutorial	103
CD	138
Selbstständigkeitserklärung	139

1. Einleitung

In der vorliegenden Arbeit wird eine zweidimensionale diskrete Fouriertransformation in VHDL implementiert. Diese Tätigkeit erfolgt im Rahmen des Projekts *Signalverarbeitung für integrated Sensor-Arrays basierend auf dem Tunnel-Magnetoresistiven Effekt für den Einsatz in der Automobilelektronik* (ISAR) an der Hochschule für angewandte Wissenschaften Hamburg.

1.1. Motivation

Bisherige Analysen und Versuche wurden überwiegend auf theoretischer Ebene mit der Simulationssoftware Matlab oder anhand einer ca. 7x7 cm großen Sensormatrix mit 64 einzelnen Sensoren in Verbindung mit einem an den PC angeschlossenen Mikrokontroller vorgenommen. Auf diese Weise können reale Messwerte ermittelt und verschiedene Szenarien getestet werden.

Das Ziel des ISAR-Projekts ist, einen Application Specific Integrated Circuit, dt.: *Anwendungsspezifischer Integrierter Schaltkreis* (ASIC) zu fertigen, auf dem die Sensoren und alle Berechnungen erfolgen. zunächst ist die Entwicklung der signalverarbeitenden Komponenten in einer Hardwarebeschreibungssprache geplant. Ziel ist ein Sensormodul, mit dem eine kontinuierliche Winkelberechnung zur Positionsbestimmung möglich ist. Dieses soll beispielsweise die Rotorlage eines Elektromotors ermitteln und die Information als digitales Nutzsignal ausgeben können. Ein wesentliches Aspekt ist das Unterdrücken von Störfeldern, welche in Form von Inhomogenitäten, unter anderem hervorgerufen durch Fehlpositionierung (Ablage) und Schrägstellung des Gebermagneten und den Magnetsensoren selbst, verursacht werden.

Den erste Schritt der Signalverarbeitung stellt die Berechnung der zweidimensionalen direkten Fouriertransformation der Sensorsignale dar (Abb. 1.1), da sich im Fourierraum Störanteile leichter filtern lassen. Derzeit ist offen, ob anschließend der Winkel direkt berechnet werden kann oder vorher eine Rücktransformation erfolgen muss.

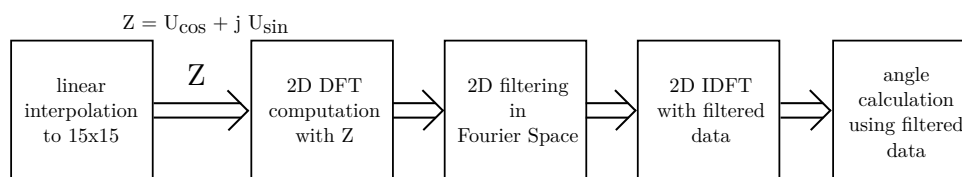


Abbildung 1.1.: Ablauf der Signalvorverarbeitung [1, S. 9]

1.2. Stand der Technik

Zur Winkelberechnung auf Basis von Magnetsensoren lassen sich verschiedene Effekte nutzen. Als erstes gelang es Sensoren auf Basis des Hall-Effekts zu entwickeln. Fortschreitende Tech-

nik ermöglichte es, die schon länger bekannten Effekte anisotropen magnetoresistiven Effekt (AMR), gigantischen magnetoresistiven Effekt (GMR) und aktuell tunneltmagnetoresistiven Effekt (TMR) herzustellen. Sensoren die den TMR-Effekt nutzen gibt es bereits seit mehreren Jahren, es werden aber weitere Fortschritte angestrebt und erzielt. Die Entwicklung von Sensoren die die verschiedenen magnetoresistive Effekte nutzen, bringen Vorteile hinsichtlich des Strombedarfs, der Amplitude des Sensorsignals, Genauigkeit der Auflösung, Größe sowie benötigte Feldstärke [2, S. 2]. Darüber hinaus ist es nur mit den GMR- und TMR-Sensoren möglich 360° aufzulösen. Die AMR-Sensoren erzeugen je Umdrehung zwei Sinusschwingungen, wodurch eine um 180° versetzte Doppeldeutigkeit entsteht [3]. Bei Hall-Sensoren sind die anderen genannten Schwächen deutlich stärker ausgeprägt, weswegen sie in diesem Bereich keine Verwendung finden.

1.3. Ziel dieser Arbeit

Um eine Aufwandsabschätzung einer 2D-DFT bezüglich Rechendauer und Flächenbedarf als Komponente auf einem ASIC zu erlangen, wird die Transformation auf ein Array angewandt, welches eine Größe hat, die sich leicht optimieren lässt. Herauszufinden, auf welche das Zutrifft, ist Teil der Arbeit. Es sollen Grundlagen erarbeitet werden, welche für die Transformation einer 15×15 -Matrix nützlich sind. Letztere wird durch lineare Interpolation der 8×8 -Sensormatrix errechnet (Abb. 1.1). Von ihr ist bekannt, dass sie wegen einer vergleichsweise hohen Anzahl verschiedener Faktoren bedeutend aufwändiger zu implementieren ist, weshalb zunächst mit einer einfacheren Berechnung Erfahrungen gesammelt und Vorarbeit geleistet werden soll. Nach Möglichkeit soll für beide Matrixmultiplikationen, die die 2D-DFT mit der Twiddlefaktormatrix erfordert, die selbe DFT-Einheit genutzt werden. Herauszufinden, ob und wie dies effizient erfolgen kann, gehört ebenfalls zur Aufgabenstellung.

2. Grundlagen

Um einen guten Ausgangspunkt für spätere Erläuterungen zu haben, werden an dieser Stelle die wesentlichen Grundlagen zusammengefasst. Dazu zählen in dieser Arbeit die Interpretation von Binär-Zahlen, die Matrixmultiplikation. Da deren Zahlen komplexwertig sein können, wird auch die komplexe Multiplikation erläutert. Abschließend werden die Berechnungen der diskreten Fouriertransformation (DFT) und der diskreten Kosinus Transformation (DCT) behandelt. Um den Zusammenhang der DFT und der Fouriertransformation sowie der Fourierreihenentwicklung als Ursprung aufzuzeigen, werden beide kurz angeschnitten.

2.1. Binäre Zahlendarstellung von Festkommazahlen

Die Eingangs- und Ausgangswerte für die Signalverarbeitung des Sensor-Arrays sollen mit einer Genauigkeit von 12 Bit aufgelöst sein. Basierend auf den bisherigen Sensoren wird von Spannungen von 0V bis 3,3V bzw. nach Abzug des Offsets $\pm 1,65V$ ausgegangen. Daraus folgt für die nachfolgenden Betrachtungen der Bereich von $-2 < z < 2$. Aus diesem Grund müssen sowohl ein Ganzzahlanteil, sowie Nachkommastellen repräsentiert werden können. Hierfür werden Festkommazahlen verwendet. Aufgrund der durchzuführenden Rechenoperationen müssen die Zielvektoren eine größere Bitbreite als die Eingangswerte haben.

2.1.1. Integer-Zahl im 1er-Komplement

Bei der Interpretation des Bitvektors als Integerwert im Einerkomplement werden die Bits anhand ihrer Position im Bitvektor gewichtet, wobei das niederwertigste Bit (LSB, least significant bit) dem Wert des Faktors 2^0 entspricht. Das Bit eine Position weiter links im Vektor entspricht dem für 2^1 und so weiter. Die Summe aller Bits, ohne das höchstwertigste, multipliziert mit ihrer Wertigkeit (Potenz) ergibt den Betrag der Dezimalzahl. Das höchstwertigste Bit (MSB, most significant bit) gibt Auskunft darüber, ob die Zahl negativ (1) oder positive (0) ist [4, S. 76]. Dies hat zur Folge, dass es eine positive und eine negative Null und somit eine Doppeldeutigkeit gibt. Daraus kann gefolgert werden, dass ein LSB an Auflösung verschenkt wird. Der Wertebereich erstreckt sich von $-2^{\text{MSB}-1} + 1 \text{ LSB}$ bis $2^{\text{MSB}-1} - 1 \text{ LSB}$.

Diese Darstellung hat den Vorteil, dass sich das Ergebnis einer Multiplikation der Zahlen $a \cdot b$ und $-a \cdot b$ nur im vordersten Bit unterscheidet. Darüber hinaus lässt sich das Vorzeichen des Ergebnisses durch eine einfache XOR-Verknüpfung der beiden MSB der Multiplikatoren ermitteln. Anhand der Wahrheitstabelle 2.1 eines XOR-Gatters [4, S. 37] ist ersichtlich, dass das Ausgangssignal X nur gesetzt ist, wenn ausschließlich eines der beiden MSB der Eingangssignale A oder B gesetzt ist. Aus dieser Tatsache leitet sich auch der Name ab, er steht für Exklusiv-Oder. Die eigentliche Multiplikation beschränkt sich auf die Bits MSB-1 bis LSB.

Nachteile zeigen sich hingegen bei der Addition und Subtraktion negativer Zahlen. Auch hierfür gibt es Berechnungsvorschriften, diese erfordern im Einerkomplement jedoch mehr Zwischenschritte als im Zweierkomplement.

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Tabelle 2.1.: Wahrheitstabelle eines XOR-Gatters. A und B sind Eingänge, X das Ausgangssignal

2.1.2. Integer-Zahl im 2er-Komplement

Vorteil bei dieser Darstellung ist, dass die mathematischen Operationen Addition, Subtraktion und Multiplikation direkt angewandt werden können. Bei der Interpretation des Bitvektors als Zweierkomplement kann anhand des MSB ebenfalls erkannt werden, ob es sich um eine positive oder negative Zahl handelt. Hier bedeutet ein gesetztes MSB -2^{MSB-1} , was der negativsten darstellbaren Zahl entspricht. Alle anderen Bits müssen auf 0 sein. Für gesetzte Bits wird der Dezimalwert, wie beim Einerkomplement beschrieben, berechnet und auf den negativen Wert aufaddiert. Wenn das MSB nicht gesetzt ist, wird der errechnete Dezimalwert auf 0 addiert. Auf diese Weise lassen sich Zahlen im Wertebereich von -2^{MSB-1} bis $2^{MSB-1} - 1$ LSB darstellen. Der positive Wertebereich ist also um ein LSB kleiner als der negative und es gibt keine doppelte Null. Um eine Zahl zu negieren, müssen alle Bits invertiert werden. Auf das Resultat muss abschließend 1 LSB addiert werden. [4, S. 76]

2.1.3. Darstellung dualer Zahlen im SQ-Format

Im SQ-Format (signed quotient) werden Zahlen als vorzeichenbehafteter Quotient dargestellt. Ein gesetztes MSB definiert wieder, dass die Zahl als negativ aufzufassen ist. In Abbildung 2.1 ist exemplarisch die Interpretation von Dualzahlen im SQ3-Format zu sehen.

Der darstellbare Zahlenbereich liegt bei $-1 \leq z < 1$. Für dieses Projekt werden Zahlen im Bereich von etwa ± 2 angenommen, weshalb ein Vorkommabit benötigt wird. Zwölf Bit stehen zur Verfügung, von denen eins für das Vorzeichen und ein weiteres für eine Vorkommastelle verwendet werden. Die zehn übrigen Bits werden für die Nachkommazahlen genutzt. Die Interpretation der Bits wird über die Bezeichnung S1Q10 definiert [4, S. 82]. Für den Quotient stehen 10 Bit zur Verfügung, weshalb die maximale Auflösung $1 \text{ LSB} = 2^{-10} = 1024^{-1} = 9,765625 \cdot 10^{-4}$ beträgt. Der Wertebereich liegt bei -2 bis $1,999\,023\,438$.

Für alle Rechenoperationen müssen die Zahlen, auf die sie angewandt wird, dieselbe Bitbreite und das gleiche Darstellungsformat besitzen.

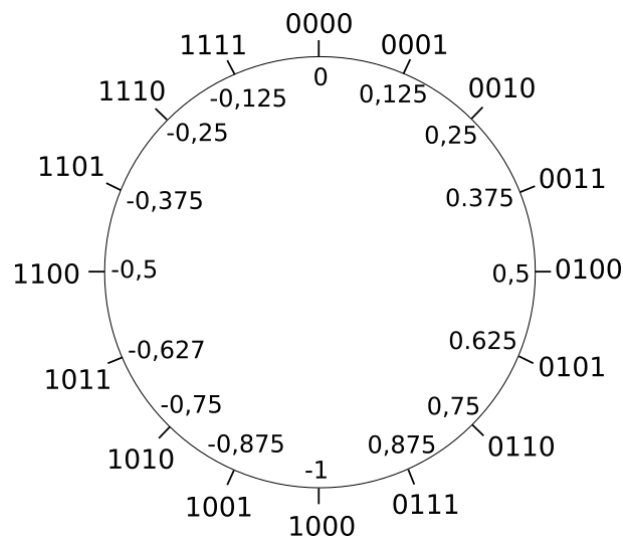


Abbildung 2.1.: Interpretation von Dualzahlen im SQ3-Format.

2.1.4. Numerische Ungenauigkeiten

Numerische Ungenauigkeiten entstehen immer dann, wenn die zur Verfügung stehenden Bits nicht ausreichen, eine Zahl exakt abzubilden. Bei einem Bitshift, welcher häufig für die Division durch zwei oder Vielfachen von zwei verwendet wird, können immer Informationen verloren gehen. Dies ist immer dann der Fall, wenn die Bits die hinausgeschoben werden eine 1 sind. Das hat zur Folge, dass beispielsweise bei einer Division durch zwei der resultierende Wert um 1 LSB kleiner ist, als er eigentlich sein sollte. Dieses Problem kann bei jedem Bitshift auftreten. Die Wahrscheinlichkeit, dass das LSB gesetzt ist, liegt im Mittel bei 50 %, weshalb davon ausgegangen werden muss, dass ein positives Ergebnis kleiner und ein negatives vom Betrag größer ist, als bei verlustfreier Berechnung.

2.2. Mathematische Grundlagen

In dieser Arbeit werden die komplexe Multiplikation sowie die Matrixmultiplikation genutzt, welche nachfolgend kurz behandelt werden. Die Fourierreihenentwicklung sowie insbesondere die Fouriertransformation und ihre diskrete Variante werden im Anschluss erläutert, da sie elementarer Bestandteil dieser Arbeit sind. Da auch die diskrete Kosinustransformation eine Möglichkeit darstellt, in den Bildbereich zu gelangen, wurde sie in den Vorüberlegungen dieser Arbeit betrachtet und wird hier gezeigt.

2.2.1. Komplexe Multiplikation

Um das Produkt einer komplexen Multiplikation zu erhalten, müssen im allgemeinen Fall gemäß Gl. (2.1) vier einfache Multiplikation, sowie zwei Additionen durchgeführt werden.

$$\begin{aligned}
e + jf &= (a + jb) \cdot (c + jd) \\
&= a \cdot c + j(a \cdot d) + j(b \cdot c) + j^2(b \cdot d) \\
&= a \cdot c - b \cdot d + j(a \cdot d + b \cdot c)
\end{aligned} \tag{2.1}$$

2.2.2. Matrixmultiplikation

Um die Abschnitte in denen die DFT als Matrixmultiplikation beschrieben wird, besser erörtern zu können, soll diese nachfolgend besprochen werden. Wie in Abbildung 2.2 verdeutlicht wird, wird $\text{Element}(i, j)$ der Ergebnismatrix dadurch berechnet, dass die Elemente (i, k) einer Zeile der 1. Matrix mit den Elementen (k, j) aus der zweiten Matrix multipliziert und die Werte aufsummiert werden. i und j sind für die Berechnung eines Elements der Ergebnismatrix konstant, während k über alle Elemente einer Zeile bzw. Spalte läuft.

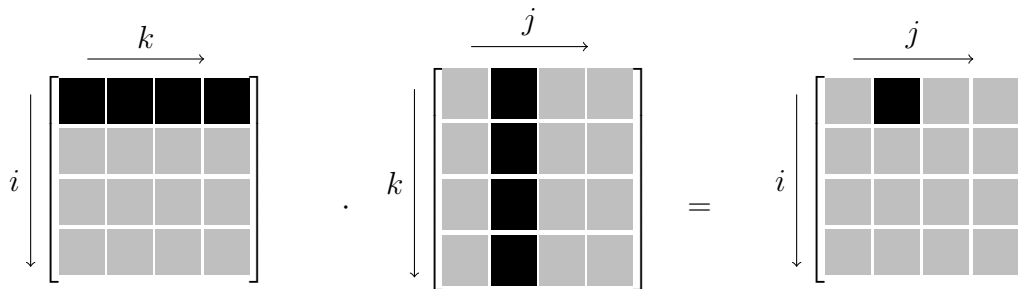


Abbildung 2.2.: Veranschaulichung der Matrixmultiplikation.

2.3. Fourierreihenentwicklung

Mit einer Fourierreihe kann ein periodisches Signal aus einer Summe von Sinus- und Kosinusfunktionen zusammengesetzt werden. Die Schreibweise als Summe von Sinus- und Kosinusfunktionen (Gl. 2.2) ist eine der häufigsten Darstellungsformen.

$$x(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos(kt) + b_k \sin(kt)) \tag{2.2}$$

Die Fourierkoeffizienten lassen sich über die Gleichungen (2.3) und (2.4) berechnen:

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} x(t) \cdot \cos(kt) dt \quad \text{für } k \geq 0 \tag{2.3}$$

$$b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} x(t) \cdot \sin(kt) dt \quad \text{für } k \geq 1 \tag{2.4}$$

Mit der Exponentialschreibweise lassen sich Sinus- und Kosinusfunktionen auch wie in (2.5) und (2.6) ausdrücken:

$$\cos(kt) = \frac{1}{2} (e^{jkt} + e^{-jkt}) \quad (2.5)$$

$$\sin(kt) = \frac{1}{2j} (e^{jkt} - e^{-jkt}) \quad (2.6)$$

Zusammengefasst ergibt (Gl. 2.7) den komplexe Zeiger, der eine Rotation im Gegenuhrzeigersinn auf dem Einheitskreis beschreibt. In Abbildung 2.3 wird dies grafisch dargestellt.

$$\begin{aligned} \cos(kt) + j \cdot \sin(kt) &= \frac{1}{2} (e^{jkt} + e^{-jkt}) + j \cdot \frac{1}{2j} (e^{jkt} - e^{-jkt}) \\ &= \frac{1}{2} (e^{jkt} + e^{jkt}) \\ &= e^{jkt} \end{aligned} \quad (2.7)$$

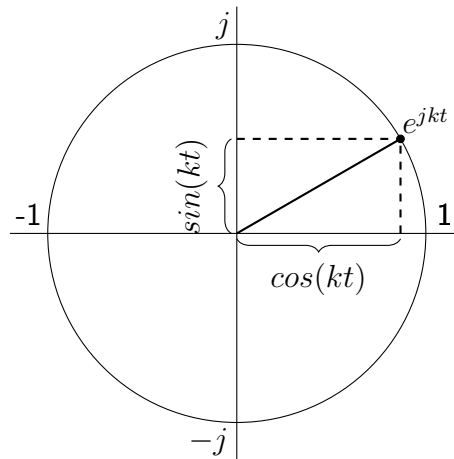


Abbildung 2.3.: Einheitskreis, Zusammensetzung des komplexen Zeigers e^{jkt} aus einer Sinus- und einer Kosinusfunktion mit dem selben Argument.

Die Fourierkoeffizienten a_k und b_k lassen sich auch als komplexe Zahl c_k zusammengefasst berechnen:

$$c_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} x(t) e^{-j2\pi kt} dt \quad \forall k \in \mathbb{Z} \quad (2.8)$$

$$x(t) = \sum_{-\infty}^{\infty} c_k e^{jkt} \quad (2.9)$$

2.4. Fouriertransformation

Mit der Fouriertransformation kann ein periodisches Signal $x(t)$ in eine Summe aus Sinus- und Kosinusfunktionen unterschiedlicher Frequenzen zerlegt werden. Da diese Funktionen jeweils mit nur einer Frequenz periodisch sind, entsprechen diese den Frequenzbestandteilen von $x(t)$.

Grundlage für die Fouriertransformation ist das Fourierintegral (Gl. 2.10)

$$X(f) = \int_{-\infty}^{\infty} x(t) \cdot e^{-j2\pi ft} \quad (2.10)$$

Wenn Sinus- und Kosinusfunktionen wie in Gl. (2.5) und (2.6) als Exponentialfunktionen geschrieben werden, können sie zu einer komplexen Exponentialfunktion zusammengefasst werden. Hieraus lässt sich ableiten, dass das Spektrum, $X(f)$ komplexwertig sein muss.

Signalformen wie etwa ein Rechteck haben entsprechend sehr viele dieser Frequenzbeiträge. Deren Wert ist Information darüber, wie groß ihr Anteil, also die Amplitude des Zeitsignals, ist. Eine Vertiefung dieses umfangreichen Gebiets der Fourier-Analyse findet sich u.a. in [5, 542 ff].

In der vorliegenden Arbeit wird X^* für die eindimensionale diskrete Fouriertransformation (1D-DFT) und X für die zweidimensionale diskrete Fouriertransformation (2D-DFT) stehen.

2.4.1. Diskrete Fouriertransformation (DFT)

Die DFT ist die zeit- und wertdiskrete Variante der Fouriertransformation, die statt von $-\infty$ bis ∞ über einen Vektor von N Werten, also von 0 bis $N-1$ läuft. Dies hat zur Folge, dass sich ihr Frequenzspektrum periodisch nach N Werten wiederholt. Da es sich um eine endliche Anzahl diskreter Werte handelt, geht das Integral aus Gleichung (2.10) in die Summe aus Gleichung (2.11) über.

Üblicherweise wird die (diskrete) Fouriertransformation genutzt, um vom Zeitbereich in den Frequenzbereich zu gelangen. In diesem Fall enthielte der Eingangsvektor Werte im Zeitbereich, der Ausgangsvektor Werte im Frequenzbereich. Um von Daten im Zeitbereich sprechen zu können, müssen diese zeitlich versetzt auf den gleichen Bezugspunkt erfasst worden sein. Bezogen auf das Sensor-Array würde eine bestimmte Anzahl an zeitlich versetzten zeit- und wertediskretisierten Daten eines einzelnen Sensors in einem Vektor zusammengefasst und darauf die DFT angewandt werden, um beim Ausgangsvektor von Daten im Frequenzbereich sprechen zu können.

Statt zeitlich versetzt, werden beim Sensor-Array die Daten von mehreren Sensoren gleichzeitig erfasst. Da das Array zweidimensional ist, handelt es sich um eine Matrix, deren Sensoren verschiedene Koordinaten repräsentieren. Aus diesem Grund muss von Orts- anstatt von Zeitwerten gesprochen werden [6, S. 69]. Von der Transformation in das Frequenzspektrum spricht man bei Zeitwerten, da das Spektrum die Frequenzen darstellt, aus denen das Zeitsignal zusammengesetzt ist. Da bei der eben beschriebenen Datenerfassung Ortsdaten transformiert werden, wird von einer Transformation in den Bildbereich gesprochen.

In dieser Arbeit werden statt Orts- und Bildbereich auch die Begriffe Eingangs- und Ausgangsvektor bzw. -matrix verwendet.

Mit der eindimensionalen diskreten Fouriertransformation (1D-DFT) wird die spaltenweise DFT einer Matrix bezeichnet, in der Regel ist sie der erste Schritt der Berechnung der 2D-DFT. Die Größe der Eingangsmatrix gibt die Größe der Twiddlefaktormatrix vor, beide müssen identisch und quadratisch sein. In dieser Arbeit wird die DFT einer Matrix der Größe $N \times N$ auch $N \times N$ -DFT genannt.

2.4.2. Summen- und Matrizenschreibweise der DFT

1D-DFT

Die DFT findet Anwendung, um vom Zeit- bzw. Ortsbereich in den Frequenz- bzw. Bildbereich zu gelangen.

$$X^*[m] = \frac{1}{N} \cdot \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi mn}{N}} \quad (2.11)$$

In Gleichung (2.11) ist die Verwendung von Eingangsvektor $x[n]$ und Ausgangsvektor $X[n]$ zu sehen. Eine spaltenweise Multiplikation einer Matrix ist auch denkbar und ist darüber hinaus Grundlage für die 2D-DFT. Gleichung (2.13) zeigt die Summenformel aus (2.11), umgeschrieben zu einer Matrixmultiplikation.

Mit Gleichung (2.12) werden alle Twiddlefaktoren in Matrixform berechnet, wobei n der Index des zu berechnenden Elements des Vektors im Zeitbereich und m das Äquivalent im Frequenzbereich ist.

$$W = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} e^{-j\frac{2\pi mn}{N}} \quad (2.12)$$

Bei genauer Betrachtung fällt auf, dass sie identisch mit ihrer Transponierten ist.

Somit gilt:

$$X^* = W \cdot x \quad (2.13)$$

Anhand der Summen die jeweils von 0 bis $N-1$ laufen, lässt sich die Anzahl der benötigten komplexen Multiplikationen m_{DFT} einer DFT errechnen. Siehe auch Gleichung (2.14).

$$m_{DFT} = N^2 \quad (2.14)$$

2D-DFT

Die 2D-DFT wird in der Bildverarbeitung verwendet, um vom Orts- in den Bildbereich zu gelangen. Da es sich nicht um eine Abhängigkeit der Zeit handelt, werden andere Indizes verwendet.

$$\begin{aligned} X[u, v] &= \frac{1}{N} \sum_{n=0}^{N-1} X^*[m] \cdot e^{-j\frac{2\pi mn}{N}} \\ &= \frac{1}{MN} \sum_{m=0}^{M-1} \left(\sum_{n=0}^{N-1} f(m, n) \cdot e^{-j\frac{2\pi mn}{N}} \right) \cdot e^{-j\frac{2\pi mn}{M}} \end{aligned} \quad (2.15)$$

Auch hier lässt sich die Berechnung in Matrixschreibweise darstellen:

$$\begin{aligned} X &= W \cdot x \cdot W \\ &= X^* \cdot W \end{aligned} \quad (2.16)$$

Die Gleichungen (2.13) und (2.16) sind in dieser Arbeit wesentlicher Bestandteil der Umsetzung der 2D-DFT.

Die 2D-DFT kann als "doppelte" Matrixmultiplikation geschrieben werden (Gleichung (2.16)). Zuerst wird die 1D-DFT berechnet und die Matrix X^* wird anschließend mit der Twiddlefaktor-Matrix W multipliziert. Man könnte es auch als zweite 1D-DFT betrachten, bei der Twiddlefaktor-Matrix und Eingangsmatrix vertauscht sind. Veranschaulicht wird dies in den Abbildungen 2.4 und 2.5.

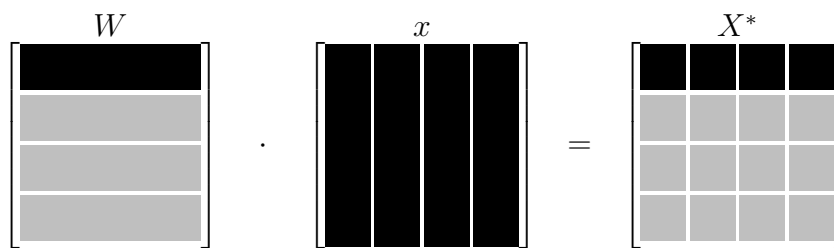


Abbildung 2.4.: 1D-DFT als Matrixmultiplikation.

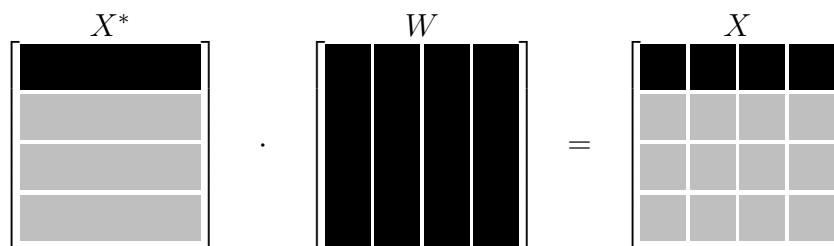


Abbildung 2.5.: 2D-DFT als Matrixmultiplikation.

2.4.3. 2D-DFT mit reellen Eingangswerten

Bei der oben beschriebenen Berechnung können die Eingangssignale auch komplex sein. Da das Ausgangssignal der 1D-DFT unabhängig von den Eingangssignalen komplex ist, kann es direkt als Eingangssignal für die komplexe 2D-DFT genutzt werden.

Das komplexe Ausgangssignal der 1D-DFT kann auch als zwei von einander unabhängige rein reelle Eingangssignale der 2D-DFTs betrachtet und später wieder zusammengesetzt werden.

werden. Gleiches gilt für ein komplexes Eingangssignal, welches in zwei von einander unabhängigen DFTs transformiert werden kann. Da bei dieser Umsetzung kein Imaginärteil in die Berechnung der Ergebnisse einfließt, hat sie den Vorteil, dass aus Symmetriegründen die Hälfte der Multiplikationen eingespart werden können. Es ist erforderlich, dass der Imaginärteil der gespiegelten Ergebnisse negiert wird. Abbildung 2.7 zeigt die redundanten Werte der DFT. Demnach müssen bei der 8x8-DFT statt 16 nur 8 Multiplikationen mit reellem Multiplikand und komplexem Multiplikator erfolgen. Anschließend müssen die Ergebnisse, wie in Abb. 2.6 zu sehen, zusammengesetzt werden. Die Abbildung stellt die schematische Berechnung der 2D-DFT eines reellen Eingangssignals dar. Um die 2D-DFT eines komplexen Eingangssignals zu berechnen, müssen bei dieser Methode zwei identische Recheneinheiten vorhanden, wenn Real- und Imaginärteil parallel berechnet werden sollen. Andernfalls müssen sie zeitlich versetzt berechnet und Zwischenergebnisse gespeichert werden. Die Ergebnisse beider 2D-DFTs müssen identisch zusammengefasst werden, wie es zum Abschluss der einzelnen 2D-DFTs geschehen muss.

Da die gegebenen Eingangssignale aus einer Sinus- und einer Kosinuskomponente bestehen und es sich auf diese Weise als ein komplexes Signal auffassen lässt, kann die komplexe Berechnung sowohl bei der 1D-DFT als auch bei der 2D-DFT genutzt werden. Da hierdurch in beiden Fällen eine vollständige Auslastung einer komplexen Berechnung gegeben ist und wie bereits erwähnt, bei der reellen Berechnung zusätzlicher Speicher erforderlich wäre, wird dieses Verfahren angewandt.

2.4.4. Berechnung der schnellen Fouriertransformation

Die Mathematiker Cooley und Tukey haben einen Algorithmus entwickelt und im Jahr 1965 veröffentlicht, mit dem sich die DFT mit weniger Multiplikationen und dadurch schneller als bei der allgemeinen DFT berechnen lässt. Das Verfahren wird als Fast Fouriertransformation (FFT) bezeichnet. Grundlage ist, dass sich eine DFT in kleinere Teil-DFTs aufspalten lässt, welche durch Ausnutzen von Symmetrieeigenschaften in der Summe weniger Koeffizienten haben. Üblich ist die Radix-2 FFT, Ausgangspunkt ist eine DFT mit 2 Eingangswerten. Diese Methode kann nur auf Eingangsvektoren der Größe 2^n angewandt werden. Dieser vermeintliche Nachteil lässt sich durch Auffüllen des Eingangsvektors mit Nullen (Zeropadding) eliminieren. Dies hat zur Folge, dass die Größe des Ausgangsvektors immer eine Potenz von zwei ist. Die Anzahl der benötigten komplexen Multiplikationen m_{FFT} kann mit der Gleichung (2.17) abgeschätzt werden.

$$m_{FFT} = \frac{N}{2} \log_2(N) \quad (2.17)$$

Das Verfahren wird in Kürze in Kapitel 3.3.4 behandelt und kann unter anderem in dem Buch *Digital Signal Processing: Principles, Algorithms and Applications* [7] auf den Seiten 511 bis 524 nachgelesen werden.

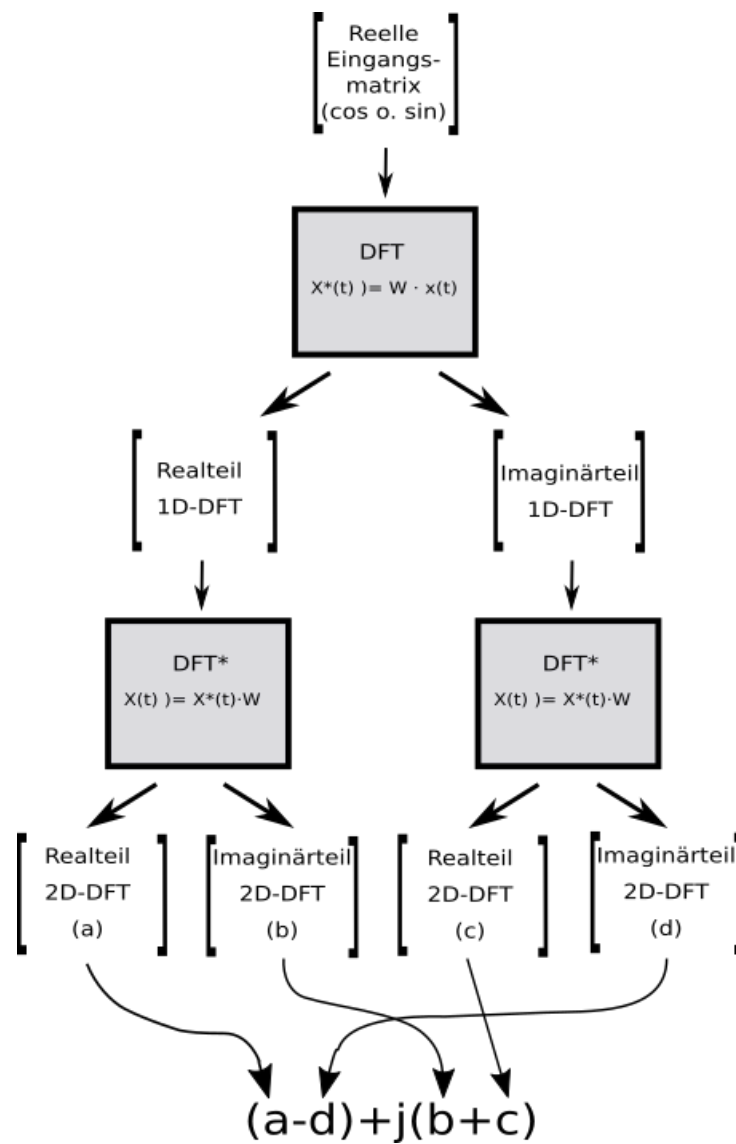


Abbildung 2.6.: Veranschaulichung der Berechnung der DFT mit reellen Eingangswerten.

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6	A4	B4	C4	D4	E4	F4	G4	H4
7	A3	B3	C3	D3	E3	F3	G3	H3
8	A2	B2	C2	D2	E2	F2	G2	H2

(a) Realteil

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6	-A4	-B4	-C4	-D4	-E4	-F4	-G4	-H4
7	-A3	-B3	-C3	-D3	-E3	-F3	-G3	-H3
8	-A2	-B2	-C2	-D2	-E2	-F2	-G2	-H2

(b) negierter Imaginärteil

Abbildung 2.7.: Redundante Werte der spaltenweisen DFT einer 8x8-Matrix. Der Imaginärteil der redundanten Werte hat denselben Betrag mit negiertem Vorzeichen.

2.4.5. Inverse DFT

Die inverse diskrete Fouriertransformation (IDFT) ist die Umkehrfunktion der DFT. Wenn das Eingangssignal $x[n]$ zeitabhängig und somit als $\vec{x}(t)$ geschrieben werden kann, dann handelt es sich bei $X^*[m]$ um dessen Darstellung im Frequenzbereich und kann als $\vec{X}^*(f)$ geschrieben werden. Mit der IDFT ist es möglich, aus der Darstellung im Frequenzbereich wieder das Zeitsignal zu errechnen [5, S. 552].

$$x[n] = \frac{1}{N} \sum_{m=0}^{N-1} X^*[m] \cdot e^{\frac{j2\pi mn}{N}} \quad (2.18)$$

Gleichung (2.18) ist bis auf die Drehrichtung des komplexen Zeigers und die Vertauschten Ein- und Ausgangsvektoren identisch zu Gleichung (2.11).

2.5. Diskrete Kosinus Transformation (DCT)

Die mit dem Dmeo-Array gewonnenen Daten lassen sich als Bild mit $8 \times 8 = 64$ Pixeln darstellen. Es ist denkbar anhand dieses Bildes die gesuchten Informationen mittels Bildverarbeitungsalgorithmen zu errechnen. Zu den häufigsten Algorithmen zählt hier die DCT, da sie aus einem reellen Eingangssignal eine ebenfalls reelle Transformierte erzeugt [8, S. 355-359]. Ein weiterer Vorteil den sie gegenüber der DFT hat, ist dass bei der periodischen Wiederholung im Spektrum Unstetigkeiten aufgrund von Kanten umgangen werden können [9, S. 28-31].

Für die Berechnung der DCT gibt es verschiedene Varianten, die sich in der Symmetrie des Spektrums unterscheiden. Die in der Bildverarbeitung häufigste ist die sogenannte *ungerade DCT* [9, S. 28-31]. Häufig wird die erste Zeile der Twiddlefaktormatrix mit dem Faktor

$1/\sqrt{2}$, sowie die gesamte Matrix mit $\sqrt{\frac{2}{N}}$, $N = \text{Anzahl Elemente in einer Zeile bzw. Spalte}$, skaliert [10]. Auf diese Weise ergibt sich eine orthogonale Matrix, bei der Inverse und Transponierte identisch sind.

Diese Variante der DCT berechnet wie folgt:

$$X^*[k] = \sum_{n=0}^{N-1} x[n] \cos \left[\frac{\pi k}{N} \left(n + \frac{1}{2} \right) \right] \quad \text{für } k = 0, \dots, N-1 \quad (2.19)$$

Wie Gleichung (2.19) entnommen werden kann, basiert die Transformation nur auf Kosinusfunktionen, weshalb auch das Spektrum rein reell ist.

3. Analyse

In diesem Kapitel werden zunächst die DFT und die DCT in verschiedenen Größen einander gegenübergestellt und eine Entscheidung darüber getroffen, welche sich besser dafür eignet, auf einem ASIC implementiert zu werden. Hierbei sind in erster Linie die Anzahl unterschiedlicher Faktoren relevant, da für identische nur eine Multiplikationseinheit erforderlich ist. Als Interessant wurden die Matrizen mit den Größen 8x8, 9x9 und 15x15 ausgewählt. Die 8x8-Matrix hat dieselbe Anzahl Faktoren, wie das derzeitige Demo-Array Sensore hat, sodass die Eingangswerte direkt transformiert werden können. Die beiden anderen haben aufgrund ihrer ungeraden Zahl ihren Mittelpunkt zwischen den mittleren Sensorelementen, was für die weitere Verarbeitung des transformierten Signals von Bedeutung ist. Die Matrix der Dimension 15x15 lässt sich durch Interpolation der Daten errechnen, während es für die 9x9-Matrix bisher keine Überlegungen gibt, wie sie errechnet werden könnte. Die 12x12, sowie die 16x16 werden zum besseren Einordnen der Bewertungen ebenfalls betrachtet. Darüber hinaus ist aus Abschnitt 2.4.4 bekannt, dass die FFT auf 2^n Elementen basiert und es sich hierbei um ein sehr schnelles und effizientes Verfahren handelt.

Im zweiten Schritt wird untersucht, wie die 8x8-?? optimiert werden kann.

3.1. Bewertung verschiedener DFT- und DCT-Größen

Damit eine gute Wahl einer geeigneten Größe für die DFT und DCT getroffen werden kann, werden im folgenden Abschnitt beide Transformationen untersucht. Die Bewertung berücksichtigt die beiden Eigenschaften Anzahl verschiedener Faktoren und die gesamte Anzahl an Faktoren, wobei die erst genannte größeren Einfluss auf eine negative Bewertung hat, da sich (betragsmäßig) gleiche Faktoren mit Hilfe des Distributivgesetzes ausklammern lassen. Bei den Faktoren wird zwischen trivialen und nicht trivialen unterschieden. Als trivial werden die Werte ± 1 , $\pm 0,5$ und 0 eingestuft, da sie durch einfache Operationen zu realisieren sind. Nicht triviale Werte wie beispielsweise $\frac{\sqrt{2}}{2}$ müssen bei einer Multiplikation voll berücksichtigt werden. Begründet wird dies mit dem dualen Zahlensystem, da eine Multiplikation mit als trivial eingestuften Werten statt eines komplexen Schaltnetzes im aufwändigsten Fall Bitshifts erfordern.

Interessant erscheint die DFT, da die Sensoren je ein Kosinus- und ein Sinussignal ausgeben, welches sich zu einem komplexen Signal zusammenfassen lässt. Der Nachteil der komplexen Ausgangsmatrix realtiviert sich dadurch deutlich. Darüberhinaus sind bei der DFT Symmetrien sowohl innerhalb der Spalten als auch der Zeilen vorhanden sind. Die DCT weist Symmetrien nur innerhalb der Zeilen auf.

3.1.1. Bewertung verschiedener DCT-Größen

In Tabelle 3.1 ist die Gegenüberstellung der genannten Größen zu sehen. Für die Bewertung wurde das Matlab-Skript aus Anhang A.1 geschrieben. Ersichtlich ist, dass die Anzahl verschiedener nicht trivialer Werte etwa der Wurzel aus der Anzahl aller Werte ist. Dies bedeutet im Umkehrschluss, dass im Schnitt jede Zeile einen neuen Faktor einführt. Die Summe nicht trivialer Werte weist bei allen Matrizen mehr als 50% auf.

Tabelle 3.1.: Bewertung der DCT-Twiddlefaktor-Matrizen

N	8	9	12	15	16
$N \times N$	64	81	144	225	256
\sum trivialer Werte	8	33	28	63	16
\sum nicht trivialer Werte	56	48	116	162	240
Anzahl verschiedener nicht trivialer Werte	7	7	10	13	15
Verhältnis \sum trivial / \sum nicht trivial	0.143	0.6875	0.2414	0.389	0.067

3.1.2. Bewertung verschiedener DFT-Größen

In der Tabelle 3.2 werden die DFT-Matrizen einander gegenüber gestellt. Anders als bei der DCT haben die Twiddlefaktormatrizen und deshalb auch das Ergebnis der DFT einen Real- und einen Imaginärteil. Die Beurteilung basiert auf dem Matlab-Skript aus Anhang A.2. Wie zu sehen ist, haben die 8x8- und die 12x12-DFT die besten Verhältnisse aus trivialen und nicht trivialen Faktoren. Die letztere wurde nur betrachtet, um mehr Matrizen miteinander vergleichen zu können. Aus diesem Grund wird die 8x8-DFT im folgenden Abschnitt genauer betrachtet.

3.1.3. Bewertungsfazit

DCT und DFT haben symmetrische Twiddlefaktormatrizen. Bei der DFT lassen sie sich in vier Viertel aufteilen, bei der DCT in zwei Hälften, weshalb bei der DFT gleiche Faktoren häufiger auftreten. Weniger verschiedene Faktoren gehen mit einer kleineren Chipfläche einher, was zusammen mit der schnellen Berechnung, also geringe Zahl benötigter Takte, die Hauptziele der Chipimplementierung darstellen. Aus diesem Grund ist unter diesem Aspekt die DFT zu präferieren.

Der Vorteil der DCT gegenüber der DFT ist, dass sie bei reellen Eingangswerten reelle Ergebniswerte liefert. Dem steht als großer Nachteil gegenüber, dass beinahe alle ihrer Twiddlefaktoren zu den nicht trivialen gezählt werden müssen. Da die Eingangswerte als komplex

Tabelle 3.2.: Bewertung der DFT-Twiddlefaktor-Matrizen

N	8	9	12	15	16
$N \times N$	64	81	144	225	256
trivial \Re	48	45	128	81	128
nicht triv. \Re	16	36	16	144	128
triv. \Im	48	21	96	45	128
nicht triv. \Im	16	60	48	180	128
\sum triv.	96	66	224	126	256
\sum nicht triv.	32	96	64	324	256
Anzahl verschiedener nicht trivialer Werte	1	7	1	13	3
Verhältnis \sum trivial / \sum nicht trivial	3	0,6875	3,5	0,3889	1

aufgefasst werden können, relativiert sich der einzige Vorteil der DCT, weshalb die DFT als Transformation gewählt wird.

3.2. Betrachtung der 8x8-DFT

In Abschnitt 3.1.2 wurde gezeigt, dass die Transformationsmatrix der 8x8-DFT die geringste Anzahl verschiedener Faktoren hat. Daran anknüpfend soll sie in diesem Abschnitt detaillierter auf ihre Eigenschaften hin untersucht. Dies wird die Grundlage für eine effiziente Implementierung sein. Die Twiddlefaktormatrix der 8x8-DFT besteht, wie bereits aus Gleichung (2.12) bekannt, aus komplexen Zeigern. Die möglichen Werte sind in Abbildung 3.1 zu sehen, in Abbildung 3.2 sind zur besseren Veranschaulichung die Zeiger auf 8 Einheitskreise aufgeteilt, wobei jeder einen Laufindex (m) des Zeitbereichs abdeckt. In den einzelnen Kreisen sind wiederum alle Laufindizes (n) des Frequenzbereichs zu sehen.

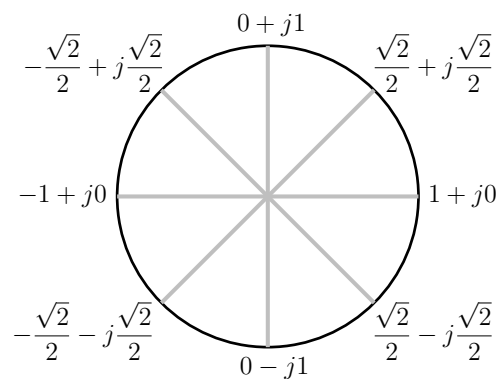


Abbildung 3.1.: Einheitskreis mit relevanten Werten der 8x8-DFT.

Wie anhand der Grafiken 3.1 zu sehen ist, setzen sich die Faktoren ausschließlich aus den

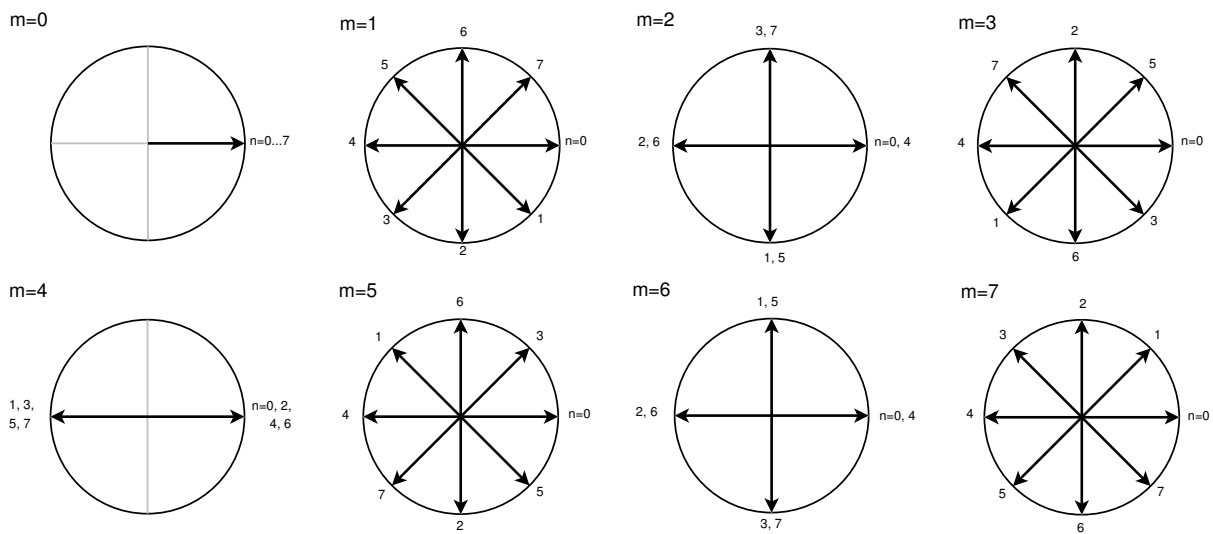


Abbildung 3.2.: Twiddlefaktoren der 8×8 -Matrix, aufgeteilt auf die Laufindizes m und n . m bezieht sich auf das Element im Ausgangsvektor \vec{X} , n auf den Eingangsvektor \vec{x} . Siehe auch Gl. (2.11).

Zahlen ± 1 , $\pm \sqrt{2}/2$ und 0 zusammen. Gemäß der Definition für nicht triviale Werte aus Abschnitt 3.1 zählt ausschließlich der letztgenannte zu diesen. Ebenfalls ist ersichtlich, dass der Betrag aller Zahlen immer 1 ist. In Abbildung 3.3 ist die Twiddlefaktormatrix auf zwei Matrizen aufgeteilt, wobei die Zahlen durch Farben repräsentiert werden. Die linke enthält alle realen Anteile, die rechte alle imaginären. Anhand der Grafik lässt sich die Symmetrie erkennen, mit der die Werte auftreten. Diese Grafik soll als Ausgangspunkt für die folgende Betrachtung dienen. Es lässt sich auch erkennen, dass die Kreise aus Abbildung 3.2 die Werte der korrespondierenden Zeilen widerspiegeln.

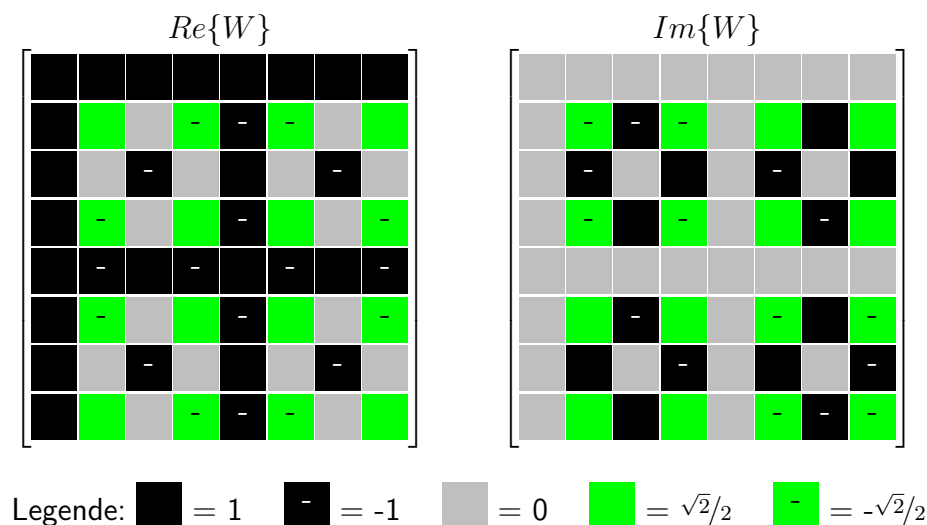


Abbildung 3.3.: Matrix-Darstellung der 8x8-DFT-Twiddlefaktoren aufgeteilt nach Real- und Imaginärteil.

In der ersten Zeile und der ersten Spalte ist der Realteil aller Elemente Eins und der Imaginärteil Null. Mit der fünften Zeile verhält es sich ähnlich. Anders ist hier, dass sich positive und negative Einsen abwechseln. In die gleiche Gruppe können noch die dritte und die siebte Zeile zusammengefasst werden. Im Unterschied zu den vorigen können hier aber auch die Imaginärteile eine positive bzw. negative Eins haben. Entsprechend ist dann der Realteil Null. Hier müssen zur Berechnung des Ergebnisses auch die Imaginärteile der Eingangsmatrix mit einbezogen werden. Für die vier bisher betrachteten Zeilen gilt, dass zur Berechnung eines Elements der Ergebnismatrix ausschließlich Additionen oder Subtraktionen erforderlich sind.

Alle Werte, die bis jetzt vorkamen, haben entweder nur einen Real- oder einen Imaginärteil. Dies hat den Vorteil, dass weniger Berechnungen erfolgen müssen, da von einer vollständig komplexen Multiplikation nur eine Multiplikation einer komplexen Zahl mit einer rein reellen (bzw. imaginären) übrig bleiben. Auf diese Weise reduziert sich der in Gleichung (2.1) gezeigte Aufwand zu dem in Gleichung (3.1). Darüber hinaus sind bei den bisherigen Zahlen keine Multiplikationen nötig, weshalb sich der Rechenaufwand auf den Additionsteil der Gleichung beschränkt.

$$\begin{aligned} e + jf &= a \cdot (c + jd) \\ &= a \cdot c + j(a \cdot d) \end{aligned} \quad (3.1)$$

Für die übrigen vier Zeilen gelten die bisherigen Beobachtungen nicht oder nur teilweise, weshalb sie nicht zur ersten Gruppe gezählt werden können. Dafür haben sie alle gemeinsam, dass die Hälfte der Faktoren sowohl einen Real- als auch einen Imaginärteil besitzen, welche symmetrisch angeordnet sind. Für diese vier Faktoren sind deshalb jeweils die gesamten vier Multiplikationen aus Gleichung (2.1) nötig.

Eine besondere Eigenschaft ist, dass der Faktor für nicht triviale Multiplikationen im Real- und Imaginärteil, zumindest vom Betrag her identisch ist. Dies liegt daran, dass der Einheitskreis in acht Teile geteilt wird und für beispielsweise $\frac{2 \cdot \pi}{8} = \frac{\pi}{4}$ der Sinus- und Kosinuswert identisch ist. Hieraus resultiert, dass die Hälfte der Berechnungen der nicht trivialen Werte, die für die reelle Matrix gemacht werden müssen, direkt für den imaginären Anteil übernommen werden kann. Die andere Hälfte muss lediglich negiert werden.

3.3. Einordnung des Rechenaufwands

Nachdem die Symmetrien der 8x8-Twiddlefaktormatrix der DFT analysiert wurden, erfolgt eine Abschätzung des Rechenaufwands. Hierbei wird in vier Kategorien unterschieden. Zum einen werden die erforderlichen Berechnungen bezüglich der 8x8-Twiddlefaktormatrix für reelle und komplexe Eingangswerte betrachtet. Als dritte Variante soll aufgezeigt werden, wie viele Multiplikationen nötig sind, wenn die Twiddlefaktormatrix als variabel angenommen würde. Als letztes soll der Butterfly-Algorithmus auf die Anzahl der benötigten Multiplikationen hin untersucht werden.

Abschließend wird die Bildung des Zweierkomplements der Konstantenmultiplikation unter dem Gesichtspunkt der benötigten Zeit und Fläche gegenüber gestellt. Dies geschieht vor dem Hintergrund, dass je nach Implementierung zwar weniger Multiplikationseinheiten, dafür aber zusätzliche Einheiten zur Negierung von Werten existieren müssen.

Da die Multiplikationen im Normalfall als bedeutend aufwändiger angenommen werden müssen, wird sich in der folgenden Betrachtung hierauf beschränkt. Tatsächlich ist es so, dass die Multiplikationen mit einer Konstanten über ein Schaltnetz innerhalb eines Taktes erfolgen können und somit bei wenigen Multiplikationen die Anzahl der Additionen an Bedeutung gewinnt. Trotzdem erlaubt der Vergleich eine gute Abschätzung.

3.3.1. 8x8-DFT mit komplexen Eingangswerten

Die Sensormatrix liefert für jedes Sensorelement einen Sinus- und einen Kosinuswert. Diese können für die Berechnung der DFT zu einer komplexen Zahl zusammengefasst werden ($\cos(x) + j \cdot \sin(x)$). Auf diese Weise lässt sich die Berechnung mathematisch kompakter durchführen.

Die Twiddlefaktormatrix der 8x8-DFT weist, wie in Abb. 3.3 zu sehen, insgesamt nur 16 Faktoren auf, die einen Real- und einen Imaginärteil besitzen. Da diese Faktoren sowohl für den Real- als auch den Imaginärteil betragsmäßig alle denselben Wert haben, lässt er sich ausklammern. Bezogen auf die erforderlichen Additionen verschiebt sich lediglich deren Durchführung auf einen früheren Zeitpunkt. Die Twiddlefaktormatrix würde somit nur noch einen einzigen komplexen Faktor in den Zeilen 2, 4, 6, 8 aufweisen. Trotz desselben Betrags beider Anteile müssen beide vorhanden sein, sonst würden die Sinusanteile keinen Einfluss in den Realteil des Ergebnisses haben. Da alle Zeilen der Twiddlefaktormatrix mit allen Spalten der Eingangsmatrix multipliziert werden müssen, ergeben sich $4 \cdot 8 = 32$ Multiplikationen für Real

bzw. Imaginärteil der Ergebnismatrix. Zusammen sind also 64 reelle Multiplikationen für die 1D- bzw. 128 für die 2D-DFT notwendig.

3.3.2. 8x8-DFT mit reellen Eingangswerten

Anders als bei der Multiplikation komplexer Eingangswerte sind bei der getrennten Berechnung von Real- und Imaginärteil ungleich viele positive und negative Faktoren je Zeile vorhanden, sodass zu diesem Zeitpunkt davon ausgegangen werden muss, dass eine Negation mancher Werte erforderlich sein wird. Wie ein Vergleich der Gleichungen (2.1) und (3.1) zeigt, entfallen die Hälfte der Multiplikationen, wenn die Eingangswerte rein reell sind. Da der Imaginärteil der Eingangswerte getrennt berechnet wird, treten diese Multiplikationen an anderer Stelle wieder auf, weshalb keine Ersparnis stattfindet. Allerdings kommen bei rein reellen Eingangswerten beispielsweise keine j^2 -Komponenten zustande, welche ausmultipliziert und anschließend addiert werden müssen. Dies führt zu der in Abschnitt 2.4.3 gezeigten Eigenschaft, dass die letzten drei Zeilen für den Realteil des Ergebnisses direkt bzw. für den Imaginärteil negiert aus den Zeilen 2-4 übernommen werden können. Spätestens an dieser Stelle müssen also Negationen erfolgen. Da zu den drei Zeilen aus Abb. 2.6 auch die beiden gehören, in denen Multiplikationen durchgeführt werden müssen, entfallen bei reellen Eingangswerten die Hälfte der Multiplikationen im Vergleich zu komplexen Eingangswerten, weshalb für die 1D-DFT nur 32 bzw. für die 2D-DFT nur 64 Multiplikationen nötig sind.

Interessant ist dieser Ansatz, wenn entweder die Recheneinheit so klein wie möglich gehalten werden soll und Real- und Imaginärteil der Eingangsmatrix nacheinander berechnet werden können oder die Berechnung äußerst schnell erfolgen muss. In beiden Fällen wird im Vergleich zur Berechnungen mit komplexen Eingangswerten deutlich mehr Speicher benötigt. Insgesamt übersteigt bei diese Art der Berechnung der Flächenbedarf der gesamten Einheit den der komplexen Variante. Auch die Leitungen um den Speicher anzubinden dürfen nicht vernachlässigt werden.

3.3.3. Direkte Multiplikation zweier 8x8 Matrizen mit komplexen Werten

Diese Art der Implementation hat den Vorteil, dass sich zu einem späteren Zeitpunkt für ein anderes Transformationsverfahren entschieden und einfach deren Twiddlefaktoren geladen werden können. Da keinerlei Optimierungen möglich sind, ist hier auch eine flexible Größe denkbar. Um einen Vergleich zu ermöglichen, wird die Multiplikation zweier 8x8 Matrizen betrachtet. Die in Abschnitt 2.2.2 erläuterte Matrixmultiplikation bedarf bei einer 8x8 Matrix je Element der Ausgangsmatrix acht komplexe Multiplikationen. Für die $8 \cdot 8 = 64$ Elemente werden deshalb 512 komplexe Multiplikationen benötigt. Da es sich sowohl bei den Eingangswerten als auch bei der Twiddlefaktormatrix um komplexe Zahlen handelt, sind, wie in Abschnitt 2.2.1 beschrieben, insgesamt $512 \cdot 4 = 2048$ Multiplikationen nötig. Für die 2D-DFT sind mit 4096 entsprechend doppelt so viele Multiplikationen nötig.

3.3.4. Betrachtung des Butterfly-Algorithmus für 8 Eingangswerte

Abbildung 3.4 illustriert die FFT anhand eines Eingangsvektors mit acht Werten. Um diesen Algorithmus anwenden zu können ist es erforderlich, dass die Werte im Eingangsvektor in umgekehrte Bitreihenfolge getauscht werden (bitreversed order). Dies geschieht nach dem Muster, dass die Indizes der Eingangswerte, wie üblich bei null beginnend, binär dargestellt werden. Nun wird die Reihenfolge der Bits getauscht. Auf diese Weise tauschen bei einem 8-Bit Vektor die Elemente 2 und 5 sowie 4 und 7 ihre Position. Andernfalls wären die Ergebnisse in vertauschter Reihenfolge. Anhand der Grafik lässt sich erkennen, dass die DFT in mehrere Stufen aufgeteilt wird.

Aus Gleichung (2.12) ist bekannt, dass die Variablen der Twiddlefaktorberechnung die Indizes der Eingangs- sowie Ausgangsvektoren sind. Hieraus lässt sich erkennen, dass die gesamte Twiddlefaktormatrix N verschiedene komplexe Werte enthält. Dies wird auch aus Abbildung 3.1 am Beispiel für $N=8$ ersichtlich. Darüber hinaus ist zu sehen, dass die komplexen Zeiger den Einheitskreis in N Bereiche mit einem Winkel von $\frac{2\pi}{N}$ unterteilen. Bekannt ist ebenfalls, dass der erste Wert immer die 1 ist. Daraus ergeben sich bei einer DFT mit zwei Eingangswerten die Twiddlefaktoren 1 und -1 , so dass eine Multiplikation entfällt. Dies bildet die erste Stufe.

Ähnlich verhält es sich mit der zweiten Stufe. Hier ergeben sich die Werte $1, -j, -1, j$, was ebenfalls bedeutet, dass keine Multiplikation erfolgen muss. Der nächste Schritt zur Reduzierung des Rechenaufwandes ergibt sich aus der Erkenntnis, dass die Werte $\exp(-i2\pi mn/N)$ und $\exp(-i2\pi \frac{mn}{2}/N) = -\exp(-i2\pi mn/N)$ ein negiertes Vorzeichen haben. Auch dies lässt sich der Abb. (3.1) entnehmen. Auf diese Weise fällt der Faktor $-j$ weg. Dies bedeutet, dass sich die Hälfte der Multiplikationen einsparen lässt.

Bei der dritten Stufe gibt es wegen der acht Eingangswerte theoretisch auch acht Faktoren. Aus den genannten Symmetriegründen halbiert sich die Anzahl. Wiederum die Hälfte davon sind komplexe Faktoren, die übrigen erfordern keine Multiplikation. Dies bedeutet, dass zwei komplexe Multiplikationen durchgeführt werden müssen, was insgesamt acht reellen Multiplikationen entspricht.

Wie gezeigt wurde, werden nur zwei komplexe Multiplikationen benötigt statt der nach Gleichung (2.17) geschätzten $8/2 \cdot 3 = 12$. So ergeben sich für alle acht Spalten einer 8×8 -Matrix nur $2 \cdot 8 = 16$ komplexe Multiplikationen. Für die 2D-DFT sind somit lediglich 32 komplexe, beziehungsweise 128 reelle Multiplikationen erforderlich.

3.3.5. Fazit der Berechnungs-Gegenüberstellungen

Es konnte gezeigt werden, dass die optimierte Matrixmultiplikation mit komplexen Eingangswerten die gleiche Anzahl benötigter Multiplikationen wie die FFT hat. Das liegt daran, dass es vom Betrag her nur einen einzigen Faktor gibt. Dieser tritt nur in der Hälfte der Zeilen der Twiddlefaktormatrix auf und kann wegen des Distributivgesetzes ausgeklammert werden. Gleiches gilt für die Berechnung mit rein reellen Eingangswerten. Hier kann wegen der Symmetrieeigenschaften zusätzlich noch die Hälfte der Multiplikationen eingespart werden. Bei Twiddlefaktormatrizen mit mehreren als nicht trivial einzustufenden Faktoren ist davon auszugehen, dass die FFT das effizientere Verfahren darstellt.

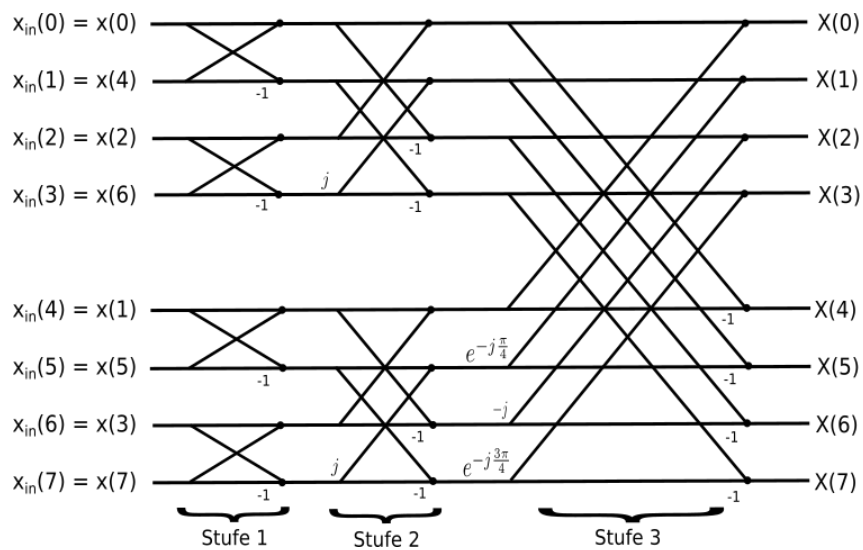


Abbildung 3.4.: Berechnungsschema der DFT mit acht Eingangswerten nach dem Butterfly-Verfahren.

Als Vorteil kann bei der Multiplikation mit komplexen Eingangswerten gesehen werden, dass die Programmierung der 2D-DFT als einfacher angenommen wird. Begründet wird dies damit, dass es möglich ist, eine einzige Einheit für die Berechnung der 1D-DFT und der 2D-DFT verwenden zu können.

Die Matrixmultiplikation mit ladbaren Faktoren ist so weit abgeschlagen, dass sie nicht ansatzweise in Betracht gezogen werden kann. Sie verdeutlicht gut, wie sehr sich Berechnungsaufwand einsparen lässt, wenn sich für ein dediziertes System entschieden und dieses optimiert wird. Falls also zwei verschiedene Transformationsverfahren auf einem Chip vorhanden sein sollen, wäre es effizienter beide optimiert zu implementieren. Abschließend werden in Tabelle 3.3 die Anzahl benötigter reeller Multiplikationen der verglichenen Methoden aufgeführt.

Tabelle 3.3.: Auflistung benötigter reeller Multiplikationen der verschiedenen Methoden für die 2D-DFT

Methode	Anzahl reeller Multiplikationen
komplexe Eingangswerte	128
reelle Eingangswerte	64
ladbare Matrixmultiplikation	4096
FFT	128

4. Entwurf

In diesem Kapitel wird die Theorie aus dem Kapitel Analyse in ein funktionsfähiges VHDL-Programm mit den geforderten Eigenschaften umgesetzt. Es werden weitere Vorüberlegungen getroffen, die sich speziell an der Implementation als Hardwarekomponente orientieren. Für eine Chipimplementation wird eine 350 nm Technologie vom Austria Microsystems AG (AMS) verwendet. Entsprechend beziehen sich die Größenangaben der Syntheseergebnisse auf diesen Prozess. Der Auftragsfertiger, bei dem der Chip in Auftrag gegeben werden würde, ist AMS. Dementsprechend sind die Standardzellen von dieser Firma.

4.1. Projekt- und Programmstruktur

In diesem Abschnitt werden Entscheidungen zum Einhalten der Bitbreite der Vektoren, den verwendeten VHDL-Bibliotheken und der Aufteilung des Programmcodes zusammengetragen.

4.1.1. Vorüberlegungen aus Hardwareasicht

Für die binäre Darstellung der Eingangswerte sind zwölf Bit vorgesehen, wovon zehn auf die Nachkommastellen entfallen. Bei den beiden Rechenoperationen Addition und Subtraktion entstehen keine weiteren Nachkommastellen. Je nach Vorzeichen und Zahlenwert kann es jedoch sein, dass der Vorkomma-Wertebereich von zwei Bit nicht mehr ausreicht. Aus diesem Grund muss der Zielvektor immer um ein Bit breiter sein - von 12 Bit ausgehend also 13 Bit. Da dies bei jeder Addition geschieht, würde die benötigte Bitbreite ohne Gegenmaßnahmen immer weiter anwachsen. Die einfachste Möglichkeit dies zu verhindern ist die Division eines Summationsergebnisses durch Zwei. Auf diese Weise kann beliebig oft auf den selben Vektor eine Addition ausgeführt werden, ohne einen Überlauf zu provozieren. Die Kehrseite dieser Vorgehensweise ist, dass durch den Bitshift die Genauigkeit des Ergebnisses sinkt. Auf die Folgen wird in Abschnitt 2.1.4 eingegangen. Für die verlustfreie Multiplikation zweier Zahlen wird die doppelte Breite des Vektors benötigt. Hier kann sich der Bitbereich vor und nach dem Komma ändern.

Wenn diese Maßnahme jedoch nicht ergriffen und alle zusätzlichen Bits beibehalten würden, müsste mit etwa 70 Bit je Ausgangswert gerechnet werden. Sollte wenigstens auf die bei der Multiplikation entstehenden Nachkommabits verzichtet werden, würde die benötigte Bitbreite immer noch bei über 40 Bit liegen. Auch dies würde noch eine immense Vergrößerung der Schaltung, alleine wegen der zusätzlichen Leitungen, bedeuten. Desweiteren würden sich hierdurch die benötigten Zeiten aller Berechnungen erhöhen, was eine langsamere Taktfrequenz oder eine Unterteilung in Teilschaltnetzte und somit in der Summe mehr Takte mit sich ziehen würde.

4.1.2. VHDL-Bibliotheken

Cadence unterstützt die VHDL-Versionen von 1987 und 1993. Enthalten ist unter anderem die Bibliothek `std_logic_arith`, welche von der Firma Synopsys entwickelt wurde. Hierbei handelt es sich um die erste Bibliothek für mathematische Berechnungen wie Addition und Multiplikation mit VHDL, weshalb sie eine große Verbreitung erlangt hat. Die Bibliothek basiert auf der vom Konsortium des Institute of Electrical and Electronics Engineers (IEEE) spezifizierten Bibliothek `std_logic`. Anders als angenommen werden könnte, handelt es sich hierbei nicht um einen offiziellen Standard. Ähnliches gilt auch für die Bibliotheken `std_logic_unsigned` und `std_logic_signed`. Leider hat Synopsys nicht konsequent die strenge Typisierung von VHDL eingehalten, weshalb es bei überladenen Funktionen möglich ist, die Datentypen `signed` und `unsigned` zu mischen, was zu einem unerwarteten Verhalten führt. Aus diesem Grund wird in diesem Projekt die Bibliothek `numeric_std` verwendet, welche einen vergleichbaren Funktionsumfang bietet, vom IEEE-Konsortium spezifiziert wurde und dieses Problem nicht aufweist. Zu ihrem Umfang gehört auch die `resize`-Funktion, welche es ermöglicht einen Vektor vorzeichengerecht zu erweitern.

Für den Standardsatz der Datentypen zu dem beispielsweise `std_logic` gehört, wird `std_logic_1164.all` verwendet. Um eine funktionale Simulation des implementierten VHDL-Quelltextes optimal testen zu können, werden die Bibliotheken `std.textio.all` und `std_logic_textio.all` eingesetzt. Diese ermöglichen das Lesen und Schreiben von Werten in eine Textdatei.

4.1.3. Vorüberlegungen zum Programmablauf

Die Parallelität der Berechnungen bestimmt auch bei gleicher Funktion der Komponenten maßgeblich die benötigten Takte der Berechnung der 2D-DFT sowie die benötigte Logik und deren Größe. Um einen guten Kompromiss aus erforderlicher Zeit und Chipfläche zu erzielen, sollen Real- und Imaginärteil die die Berechnung der Matrixelemente jeweils gleichzeitig, die einzelnen Elemente aber nacheinander berechnet werden. Darüber hinaus ist geplant, die selbe Recheneinheit der 1D-DFT auch für die 2D-DFT zu nutzen.

4.1.4. Struktureller Aufbau

Das Programm wurde, wie bei umfangreicheren Projekten üblich, auf verschiedene Dateien aufgeteilt. Alle Konstanten werden in einem Paket deklariert, welches in allen anderen Dateien eingebunden werden muss. So ist es möglich z.B. die Bitbreiten eines Vektors an einer zentralen Stelle zu setzen. Diese liegen für Eingangswerte bei 12, Summen 13 und Produkte 26 Bit. Da alle anderen Dateien auf diese Konstanten zugreifen, muss dieses Paket zuerst kompiliert werden. In einem weiteren Paket findet die Deklaration der eigenen Datentypen statt. Hier sei insbesondere die 8x8 Matrix mit ihren 12 Bit Vektoren vom Typ `signed` erwähnt, welche für die eingelesenen Daten, die Zwischenergebnisse (1D-DFT) sowie die Ausgangswerte verwendet wird. Da die weiteren Dateien diese Datentypen verwenden, ist es erforderlich, dass dieses Paket als zweites kompiliert wird. Alle weiteren Pakete können in beliebiger Reihenfolge kompiliert werden, da sie erst später ineinander greifen. Zum Testen bzw. für die Simulation

müssen Eingangswerte geladen werden. Hierfür wurde die Komponente `read_input_matrix` geschrieben. Die Werte müssen in einer Datei mit dem Namen `InputMatrix_komplex.txt` stehen und im dualen Zahlenformat vorliegen. Die Datei besteht aus 16 Spalten und acht Zeilen, in den leerzeichengetrennten Spalten stehen immer im Wechsel der Real- und der Imaginärteil einer Zahl. Die berechneten Ergebnisse werden mit der Komponente `write_results` im gleichen Format in eine Datei geschrieben, wie sie in der Input-Datei stehen.

4.2. Entwicklung der 2D-DFT-Komponente

Bis die Berechnung der 2D-DFT realisiert war, wurden verschiedene Stadien durchlaufen. Im ersten Schritt wurde die 1D-DFT implementiert, wobei die im Kapitel Analyse behandelten Optimierungsmöglichkeiten näher betrachtet werden. Desweiteren wird das Berechnungsschema der geraden sowie ungeraden Zeilen der Twiddlefaktormatrix und die daraus resultierende Anzahl an Takten vorgestellt. Ein wichtiger Punkt ist die Umsetzung der 2D-DFT auf Basis der vorhandenen 1D-DFT-Einheit. Abschließend wird gezeigt, dass es auf einfache Weise möglich ist, die vorhandene DFT-Einheit um die Funktion der IDFT zu ergänzen.

4.2.1. Optimierte 8x8 DFT als Matrixmultiplikation

Anfangs wurde angenommen, dass Multiplikationen mit den Twiddlefaktoren ± 1 und $\pm \frac{\sqrt{2}}{2}$ durchgeführt werden müssen. Dass bei einer optimierten 8x8-DFT wegen des expliziten ausprogrammierens der Berechnungen die Multiplikation mit ± 1 wegfällt, war schnell klar. Wegen der betragsmäßig identischen nicht trivialen Twiddlefaktoren wurde zu Beginn der Entwicklung in Betracht gezogen das 1er-Komplement zu verwenden, da sich negative und positive Zahlen mit gleichem Betrag nur durch ihr höchstwertigstes Bit unterscheiden. Auf diese Weise könnte dasselbe Resultat für den Imaginär- wie für den Realteil verwendet werden. Das Vorzeichen würde sich über eine einfache XOR-Verknüpfung beider MSB der Multiplikanden ergeben. Diesem Vorteil steht jedoch eine aufwändigere Subtraktion (bzw. Addition negativer Zahlen) gegenüber. Der zusätzliche Aufwand entspricht etwa dem der Negierung von Zahlen im 2er-Komplement. Aus diesem Grund wurde sich hierfür entschieden.

Bei der genaueren Betrachtung der Twiddlefaktormatrix konnte in Abschnitt 3.3.1 festgestellt werden, dass in jeder Zeile, abgesehen von der ersten, gleich viele Additionen wie Subtraktionen vorhanden sind. Dies trifft auch ausschließlich auf die jeweils vier nicht trivialen Faktoren der geraden Zeilen zu. Dies lässt sich anhand des Einheitskreises in Abb. 3.1, der Abbildung 3.2 und der grafischen Darstellung der Twiddlefaktormatrix in Abb. 3.3 nachvollziehen. Darüber hinaus ist ersichtlich, dass für komplexe Eingangswerte in den Zeilen 2, 4, 6 und 8 zwölf und in den übrigen acht Multiplikationen erfolgen müssen. Dies kann anhand der Gleichungen (2.1) und (3.1) nachvollzogen werden.

Aus Abschnitt 2.2.2 ist bekannt, dass bei einer Matrixmultiplikation Elemente multipliziert und anschließend die Ergebnisse aufsummiert werden. Hieraus kann abgeleitet werden, dass es das Assoziativgesetz erlaubt die Eingangswerte umzusortieren, wenn auch die Twiddlefaktoren entsprechend umsortiert werden. Geschickt aufgeteilt auf triviale und nicht triviale Berechnun-

gen, ist es dadurch möglich auf das Invertieren der Eingangswerte, also den hierfür benötigten Takt und die Inverter zu verzichten und um nur noch die Multiplikation mit $+\frac{\sqrt{2}}{2}$ durchführen zu müssen. Letztere muss wegen des Distributivgesetzes nur ein mal pro Zeile und Real- bzw. Imaginärteil erfolgen.

Darüber hinaus minimiert sich bei dieser Anordnung das Risiko eines Überlaufs, da zumindest im ersten Schritt eine Subtraktion erfolgt. In den weiteren Berechnungen folgt dann die Akkumulation. Wegen der Einsen in der ersten Zeile der Twiddlefaktormatrix müssen für die erste Zeile der Ausgangsmatrix alle Spalten einmal aufsummiert werden. Dies hat zur Folge, dass ein großer Wert entstehen kann, welcher Maßgebend für die Anzahl der Vorkommabits ist. Aus diesem Grund wird zur Sicherheit, der einheitlichen Skalierung der Zahlen und der Einfachheit wegen nach jeder Addition oder Subtraktion das Ergebnis durch einen Bitshift halbiert. Es sei an dieser Stelle lediglich angemerkt, dass über die Eingangswerte die Annahme getroffen werden kann, dass aufeinanderfolgende Werte das selbe Vorzeichen haben oder, nach Abzug eines evtl. vorhandenen Offsets, nahe Null sind. Basierend auf diesem Wissen ist es denkbar die Wahrscheinlichkeit weiter zu reduzieren, dass es zu einem Überlauf kommt. Wegen der Null im Imaginärteil der ersten und fünften Zeile der Twiddlefaktormatrix sind auch alle Imaginärteile dieser Spalte der Ausgangsmatrix Null. Wie in Abschnitt 2.1.4 erwähnt, ist die Optimierung bezüglich numerischer Ungenauigkeiten nicht Gegenstand dieser Arbeit.

4.2.2. Berechnungsschema und benötigte Takte der Ergebnisse

In Abbildung 4.1 ist die Berechnung der ungeraden Spalten der Eingangsmatrix am Beispiel der ersten Spalte zu sehen. Für die 3. und 7. müssen die Eingangswerte so angeordnet werden, dass die Vorzeichen, beginnend mit einer Subtraktion, immer im Wechsel auftreten. Für die 5. Zeile ist dies bereits gegeben. r steht für den Realteil der Eingangswerte, i stünde für dessen Imaginärteil. Der Index gibt die Position des Elements in einer Spalte an, angewandt werden muss die Berechnung auf alle Spalten.

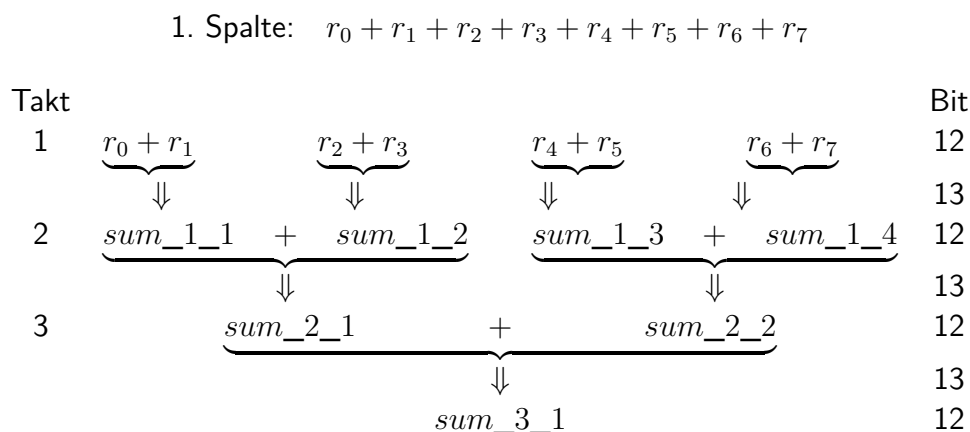


Abbildung 4.1.: Vorgehensweise der Akkumulation der ungeraden Spalten der Eingangswerte.

Wie der linken Spalte der Abb. 4.1 zu entnehmen ist, werden 3 Takte für die Berechnungen

der Werte aus den ungeraden Spalten der Eingangsmatrix benötigt. Der 1. Takt für Additionen bzw. Subtraktionen und 2. sowie 3. Takt für das Aufsummieren. Der Bitvektor des Ergebnisses ist zwar 12 Bit breit, aber beim letzten Bitshift von 13 auf 12 werden nur 11 Bit übernommen. Es wird also ein doppelter Bitshift vollzogen. Dies erfolgt, damit sowohl in den geraden als auch in den ungeraden Zeilen gleich viele Bitshifts erfolgen und die Werte somit identisch skaliert sind.

Die Berechnung der geraden Zeilen wird in Abbildung 4.2 am Beispiel der zweiten Zeile gezeigt. r steht wieder für den Realteil der Eingangswerte, i für dessen Imaginärteil. Auch hier ist der linken Spalte die Anzahl der benötigten Takte zu entnehmen. In diesem Fall dauert die Berechnung 5 Takte. Diese setzen sich zusammen aus ein Takt für Additionen bzw. Subtraktionen, 2.-3. sowie 5. Takt für das Aufsummieren und der 4. Takt für die Multiplikationen. In Abschnitt 4.3.1 wird gezeigt, dass die Multiplikation mit einer Konstanten innerhalb eines Taktes mit einem Schaltnetz erfolgen kann.

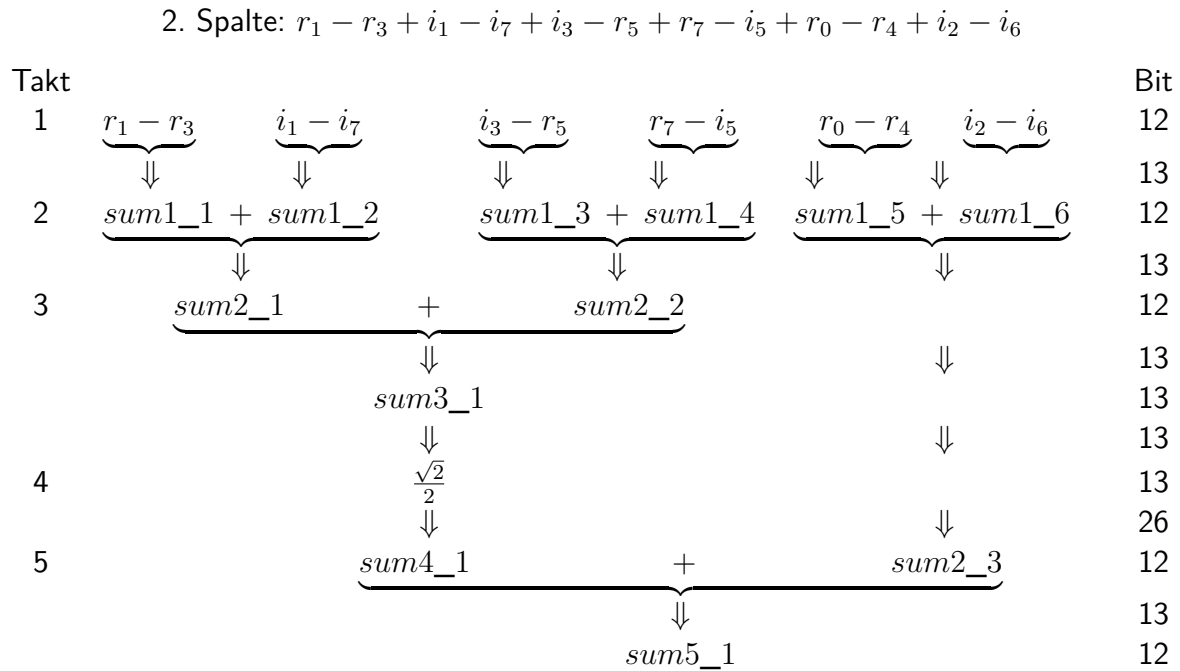


Abbildung 4.2.: Vorgehensweise der Akkumulation und Multiplikation der geraden Spalten der Eingangswerte.

Der rechten Spalte aus Abb. 4.2 kann entnommen werden, dass sich durch die Addition eine Bitbreitenerweiterung um eins bzw. bei der Multiplikation eine Verdoppelung ergibt. Durch die hintereinander erfolgenden Bitshifts wird durch 2^{n_B} geteilt, wobei n_B die Anzahl der Bitshifts ist. Um einen Überlauf zu verhindern, wird nach jeder Summation ein Bitshift durchgeführt. Da in den ungeraden Spalten weniger Summationen notwendig sind, erfolgt abschließend ein doppelter Bitshift. Andernfalls wären die Ergebnisse nicht mit denen der geraden Zeilen vergleichbar. Auf diese Weise ergibt sich für die 1D-DFT, dass das Ergebnis um den Faktor $2^4 = 16$ kleiner ist. Bei der zweiten Matrixmultiplikation wird die 2D-DFT berechnet. Auch hier wird durch 16 geteilt. Insgesamt ergibt sich eine Division durch $2^{2 \cdot 4} = 256$.

Aus den Abbildungen 4.1 und 4.2 können die Takte, die zur Berechnung der 1D- bzw.

2D-DFT benötigt werden, abgeleitet werden. Für ungeraden Zeilen sind je Element drei Takte nötig und mit acht Elementen pro Zeile und vier ungeraden Zeilen errechnen sich so 96 ($3 \cdot 8 \cdot 4$) Takte. Analog errechnet sich für die geraden Zeilen mit je fünf Takten pro Element 160 ($5 \cdot 8 \cdot 4$) Takte. In Summe ergeben sich 256 ($96 + 160$) Takte für die 1D-DFT. Die 2D-DFT kann ohne Umspeichern berechnet werden, weshalb keine weiteren Takte benötigt werden als die für die zweite Matrixmultiplikation. Deshalb verdoppelt sich die Anzahl der Takte für die vollständige Berechnung auf 512.

Tabelle 4.1.: Benötigte Takte für die komplexe DFT

Zeile	Additionen pro Element (N)	Takte pro Element ($\log_2(N)$)	Takte für Multiplikation	Summe der Takte
1	8	3	0	3
2	12	3,6	1	5
3	8	3	0	3
4	12	3,6	1	5
5	8	3	0	3
6	12	3,6	1	5
7	8	3	0	3
8	12	3,6	1	5

Anhand der rechten Spalte ergeben sich so $(3+5) \cdot 4 \cdot 8 = 256$ Takte sowohl für den Real- als auch den Imaginärteil der komplexen Ausgangsmatrix. Real- und Imaginärteil werden parallel berechnet und sind somit zeitgleich fertig.

4.2.3. Programmablauf der 1D-DFT

Der Programmablauf der 1D-DFT teilt sich für die ungeraden in drei bzw. für die geraden in fünf Takte für die Berechnung eines Elements der Ausgangsmatrix. Im ersten Takt der Berechnung werden die für die jeweilige Zeile der Twiddlefaktormatrix spezifischen Additionen und Subtraktionen durchgeführt. Für die geraden Zeilen werden die Werte, die eine Multiplikation benötigen, getrennt von den übrigen zusammengefasst. Im darauffolgenden Takt werden die Zwischenwerte, für die geraden Zeilen wieder getrennt nach Multiplikation oder nicht, akkumuliert. Der dritte Takt ist für die ungeraden Zeilen der letzte, da nur acht Werte aufsummiert werden müssen (siehe Abb. 4.1). Im darauffolgenden Takt wird in den ungeraden Zeilen mit der Berechnung der nächsten Zahl begonnen. Die geraden Zeilen haben acht Werte, die mit $\sqrt{2}/2$ multipliziert werden müssen. Die vier Werte, die nur addiert werden müssen, sind bereits im zweiten Takt aufsummiert. Deshalb kann im vierten Takt die Konstantenmultiplikation erfolgen und abschließend im fünften die Summation der Zwischenergebnisse erfolgen (siehe Abb. 4.2).

Die Zuordnung der aktuellen Zeile der Twiddlefaktormatrix erfolgt über einen Zähler, der von 0 bis 63 zählt. Er ist als 6 Bit Vektor realisiert, der bei 63 einen beabsichtigten Überlauf hat und wieder bei 0 anfängt. Der Vektor kann als zwei aufeinanderfolgende 3 Bit Vektoren gesehen werden. Auf den vorderen wird immer eine Eins aufaddiert, wenn der hintere

einen Überlauf hat und wieder bei Null zu zählen beginnt. Beide Vektoren können für sich als Modulo-8-Zähler betrachtet werden. Die vorderen drei Bit des gesamten Vektors entsprechen der aktuellen Zeile, die hinteren drei Bit der Spalte, also dem Index in der aktuellen Zeile. Mit der Funktion `to_integer` der VHDL-Bibliothek `numeric_std` lassen sich die Teilvektoren nutzen, um die Elemente der Eingangsmatrix anzusprechen. Das letzte Bit des ersten Teilvektors ist für ungerade Zeilen Null und für gerade Zeilen Eins. Da sich die Eigenschaften der Twiddlefaktormatrix in gerade und ungerade Zeilen aufteilen lassen, kann dies genutzt werden, um die entsprechenden Operationen durchzuführen und den passenden Folgezustand des Zustandsautomaten zu setzen.

4.2.4. Entwicklung von der 1D-DFT zur 2D-DFT

In Abschnitt 4.1.3 wurde entschieden, dass nach Möglichkeit die selbe Recheneinheit für die 1D- und die 2D-DFT genutzt werden soll. Aus Abschnitt 2.4.2 ist bekannt, dass für die 2D-DFT einer Matrix die Twiddlefaktormatrix einmal links und einmal rechts von der Eingangsmatrix steht. Als 1D-DFT-Matrix (Zwischenwertematrix) wird die Multiplikation der ersten Twiddlefaktormatrix mit der Eingangsmatrix betrachtet. Anschließend wird die 1D-DFT-Matrix mit der Twiddlefaktormatrix multipliziert. Aus der vertauschten Reihenfolge resultiert, dass die Zeilen und Spalten getauscht durchlaufen werden. Da die Eingangsmatrix spaltenweise durchlaufen werden soll, müsste für jedes Element eine aus Hardwaresicht aufwändige Fallunterscheidung erfolgen. Umgehen lässt sich das, indem das Kommutativgesetz angewandt wird und die Matrizen transponiert werden. Belegt wird das mit der Gleichung (4.1). Wie zu sehen ist, muss auch die Ausgangsmatrix transponiert werden. Für die Twiddlefaktormatrix erübrigt sich dies, da ihre Transponierte identisch ist. Es konnte gezeigt werden, dass beide Berechnungen mit der selben Einheit durchführbar sind. In Grafik (4.3) ist das beschriebene veranschaulicht. Das Transponieren der Zwischen- und der Ausgangsmatrix erfolgt mit vertauschtem Zeilen- und Spaltenindex. Für die Unterscheidung zwischen 1D- und 2D-DFT wird ein Bit-Signal getoggelt, wenn der in Abschnitt 4.2.3 eingeführte Vektor bis 63 gezählt hat.

$$\begin{aligned}
 X &= W \cdot x \cdot W \\
 &= \left((x \cdot W)^T \cdot W \right)^T \\
 &= \left(X^{*T} \cdot W \right)^T
 \end{aligned} \tag{4.1}$$

Da die Matrizen transponiert werden, ist eine zweite interne Matrix erforderlich, da sonst die Werte oberhalb der Nebendiagonalen überschrieben werden.

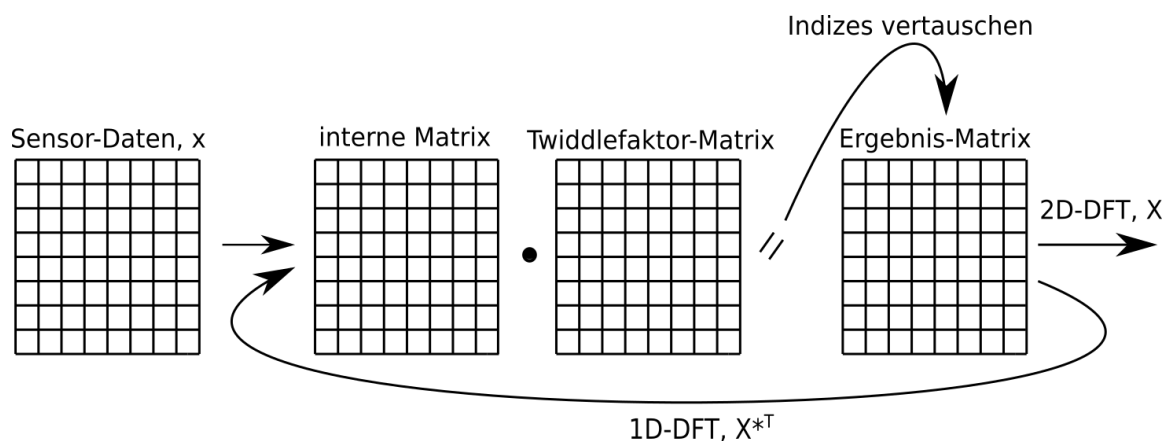


Abbildung 4.3.: Darstellung der Berechnung der 2D-DFT aus Gleichung (4.1).

4.2.5. Zusammenhang von DFT und IDFT bei der Matrixmultiplikation

Durch die umgekehrte Drehrichtung des komplexen Zeigers in Gleichung (2.18) werden in der Matrizenschreibweise die Zeilen 2 und 8, 3 und 7 sowie 4 und 6 vertauscht. Nachvollziehen lässt sich das gut anhand der Grafik 3.1. Verdeutlicht wird das vorgehen in Abbildung 4.4.

Dies lässt sich nutzen, um auf einfache Weise die vorhandene DFT-Einheit um die IDFT-Funktionalität zu ergänzen. Die DFT-Komponente wird dafür um das Eingangssignal `idft` ergänzt. Im Quelltext erfolgt eine Abfrage, ob `idft` gesetzt ist. Ist dies der Fall, wird der in Abschnitt 4.2.3 eingeführten Teilvektor für das Durchlaufen der Zeilen in seiner Integer-Interpretation von Acht subtrahiert.

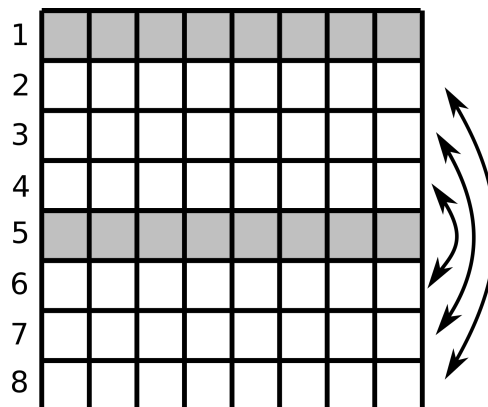


Abbildung 4.4.: Vertauschen der Zeilen 2 und 8, 3 und 7 sowie 4 und 6 der Twiddlefaktormatrix, um von der DFT zur IDFT zu gelangen.

4.3. Syntheseergebnisse von Teilkomponenten

Die Syntheseergebnisse werden mit dem Programm RTL Compiler (rc) erstellt. Wenn das Schaltnetz der gesamten Schaltung zu groß und nicht nachvollziehbar wäre, werden in diesem Abschnitt nur relevante Teilkomponenten gezeigt.

4.3.1. 13 Bit Konstantenmultiplizierer

Der duale Wert lässt sich am einfachsten mit der Matlab-Funktion `fi()` ermitteln. Der Funktion werden hierfür Kommagetrennter Dezimalwert, 1 für vorzeichenbehaftet, die gesamte Anzahl an Stellen (13) und die Anzahl der Nachkommastellen (10) übergeben. Der vollständige Aufruf sieht dann wie folgt aus:

```
val=fi(sqrt(2)/2,1,13,10)
```

Der erzeugte Datentyp hat unter anderem die Eigenschaften `val.bin`, welche einem mit 0001011010100 den Wert als Binärzahl zurück gibt, `val.double` gibt den approximierten Dezimalwert mit 0,70703125 zurück und `val.dec` interpretiert den Dualwert als Integer, was 724 entspricht. Letzterer ist wichtig zu kennen, um die Werte der Simulation nachvollziehen zu können.

Der Berechnung aus Gleichung (4.2) kann entnommen werden, dass die Abweichung weit unter einem Prozent liegt.

$$\frac{\frac{100}{\sqrt{2}}}{2} \cdot 0,70703125 = 99,989\% \quad (4.2)$$

Zeigen, welche Bits heraus genommen werden müssen! und belegen warum.

Der vollständige Gate-Report befindet sich in Abschnitt B.1 auf Seite 69

4.3.2. Bildung des 2er-Komplements eines 13 Bit Vektors

In Abb. 4.6 ist die nicht explizit implementierte, aber in Abschnitt 3.3.2 erwähnte Negierung von Zahlen zu sehen.

Für die Negierung eines 13 Bit Vektors hat das Synthesewerkzeug encounter 22 Standardzellen verwendet. Das sind knapp doppelt so viele Gatter, wie der Vektor Bits breit ist. Der Unterschied zum Konstantenmultiplizierer fällt somit sehr gering aus. Wie zu sehen, handelt es sich fast ausschließlich um Inverter und Addierer. In Abschnitt 2.1.2 wurde bereits beschrieben, dass für die Bildung des 2er-Komplements zunächst alle Bits invertiert werden müssen. Abschließend wird auf den Vektor 1 LSB addiert. Beide Pfade weisen die gleiche Länge auf und verwenden überwiegend die selben Gattertypen, weshalb darauf geschlossen werden kann, dass die maximale Gatterlaufzeit in der gleichen Größenordnung liegen muss.

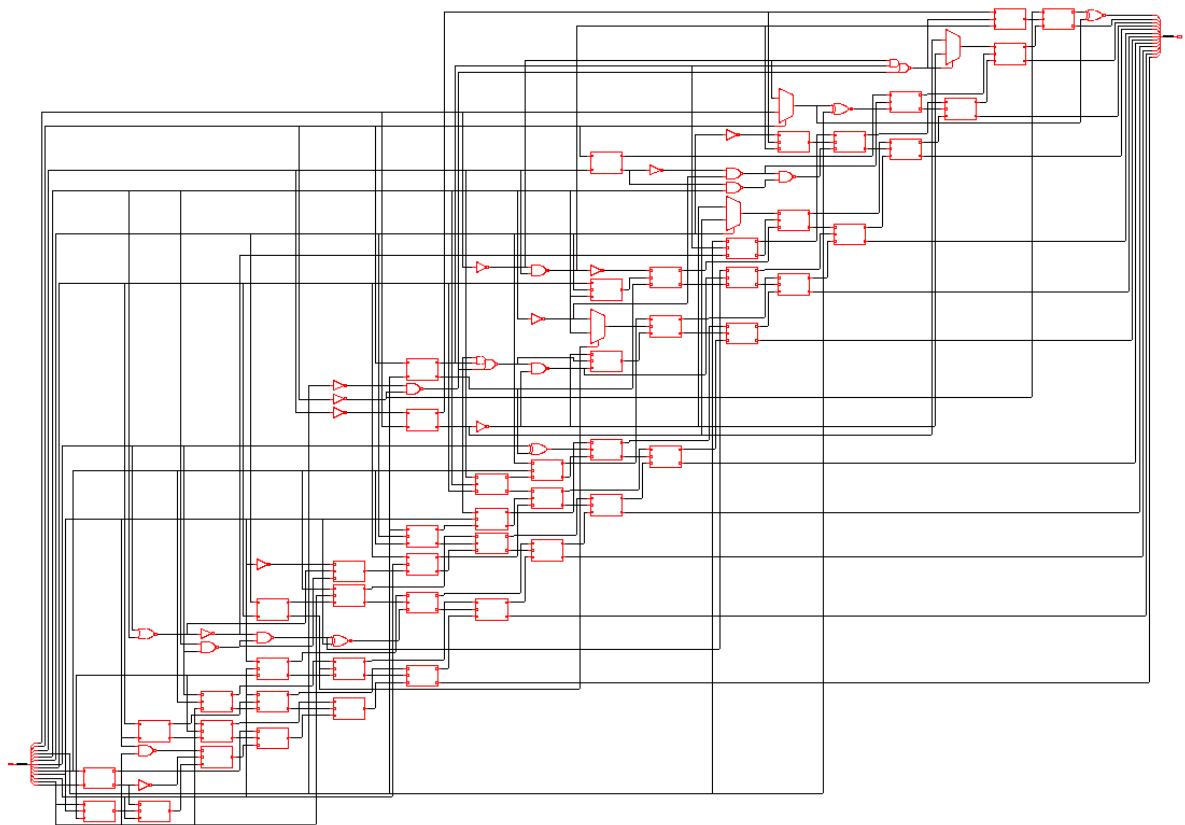


Abbildung 4.5.: Schaltnetz des 13 Bit Konstantenmultiplizierers für $\frac{\sqrt{2}}{2} = 0.70711 \simeq 0.70703125 = 0001011010100_2$ in Encounter; Eingang links, Ausgang rechts.

4.3.3. 13 Bit Addierer

Der 13 Bit Addierer hat zwei 13 Bit Eingänge, allerdings werden durch einen Bitshift beide Eingangswerte halbiert, damit kein Überlauf entsteht. Insofern fließen nur jeweils die höheren 12 Bit in die Berechnung ein.

4.3.4. Vergleich der Synthesergebnisse

In Tabelle 4.2 ist eine Gegenüberstellung der Synthesergebnisse unter den Aspekten Anzahl der Gatter und Fläche.

Tabelle 4.2.: Vergleich der Synthesergebnisse der Teilkomponenten in Bezug auf Anzahl der Gatter und ihre Fläche

	Gatter	Fläche (Prozess: 350nm)
13 Bit Konstantenmultiplizierer für $\sqrt{2}/2$	82	6612 μm^2
13 Bit regulärer Multiplizierer	194	21 985 μm^2
12 Bit Addierer	15	3257 μm^2
13 Bit 2er-Komplement-Bildung	24	2147 μm^2

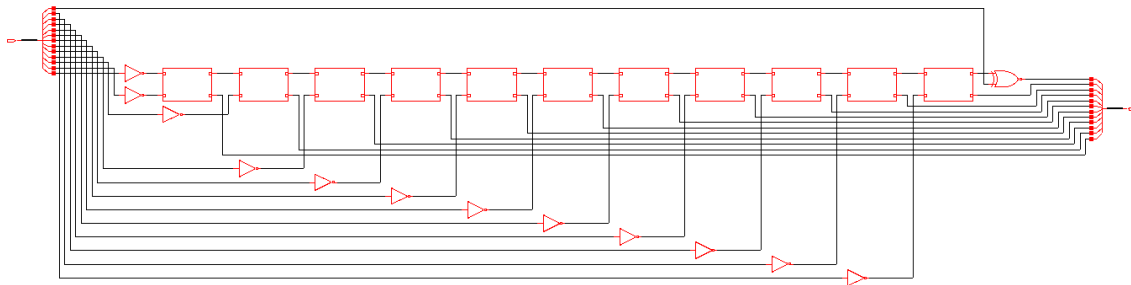


Abbildung 4.6.: Schaltnetz einer Einheit zur Bildung des 2er-Komplements eines 13 Bit Vektors; Eingang links, Ausgang rechts.

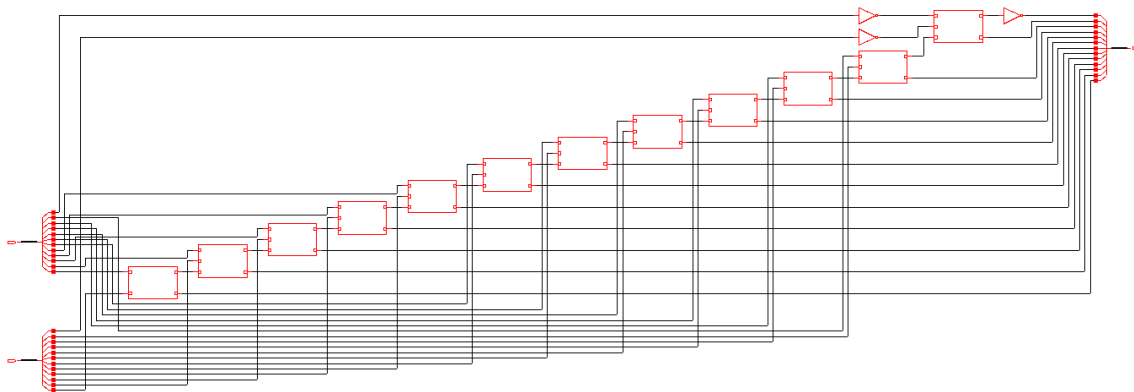


Abbildung 4.7.: Schaltnetz eines 12 Bit Addierers, Eingänge links (12 Bit), Ausgang rechts (13 Bit).

4.3.5. Gegenüberstellung der Konstantenmultiplikation und der Bildung des 2er-Komplements

Unter diesem Punkt werden die Konstantenmultiplikation und die Bildung des 2er-Komplements unter Aspekten der benötigten Zeit und des benötigten Platzes auf einem Chip betrachtet. Um einen Eindruck hiervon zu erhalten, werden im Kapitel Entwurf in Abschnitt 4.3 jeweils die Schaltnetzze gezeigt. Wie erläutert, kann gesagt werden, dass es bei dieser Art der Implementierung keinen zeitlichen Gewinn bezüglich der Taktfrequenz gibt, da beide kritischen Pfade etwa gleich lang sind. Für die knapp 4 mal mehr Gatter bei der Multiplikation ist auch ein größerer Verdrahtungsaufwand erforderlich, sodass die Konstantenmultiplizierer auf einem Chip eine etwas größere Fläche beanspruchen. Da es sich hier insgesamt aber um wenige Gatter handelt, wirkt sich dies erst bei sehr vielen Instanzen aus. Es kann an dieser Stelle festgehalten werden, dass dieser Unterschied nicht als entscheidend geltend gemacht werden kann.

4.4. Schema der Zustandsfolge

In Abbildung 4.8 sind die Zustände des in VHDL implementierten Zustandsautomaten zu sehen. Das Programm hat einen sehr strikten Ablauf und kommt ohne Eingangssignale aus, die Einfluss auf die Zustandsfolge haben, weshalb dem Automatengraf nicht viele Informationen entnommen werden können. Aus diesem Grund ist in Abschnitt 4.5 ergänzend das UML-Diagramm mit der detaillierten Abfolge der Berechnung zu sehen, welches einen wesentlich höheren Informationsgehalt besitzt. Es wurde so gestaltet, dass der zum Programmablauf gehörende Zustand ersichtlich ist.

Der Zustand `set_ready_bit` wird nach dem zweiten Durchlauf der DFT-Einheit, also wenn die 2D-DFT berechnet ist, erreicht. Dieser setzt für einen Takt das Signal `result_ready` auf Eins. Wenn `element(3)=0` ist, ist es eine ungerade Zeile der Twiddlefaktormatrix. `dft_1d_2d=0` besagt, dass die 1D-DFT berechnet wird.

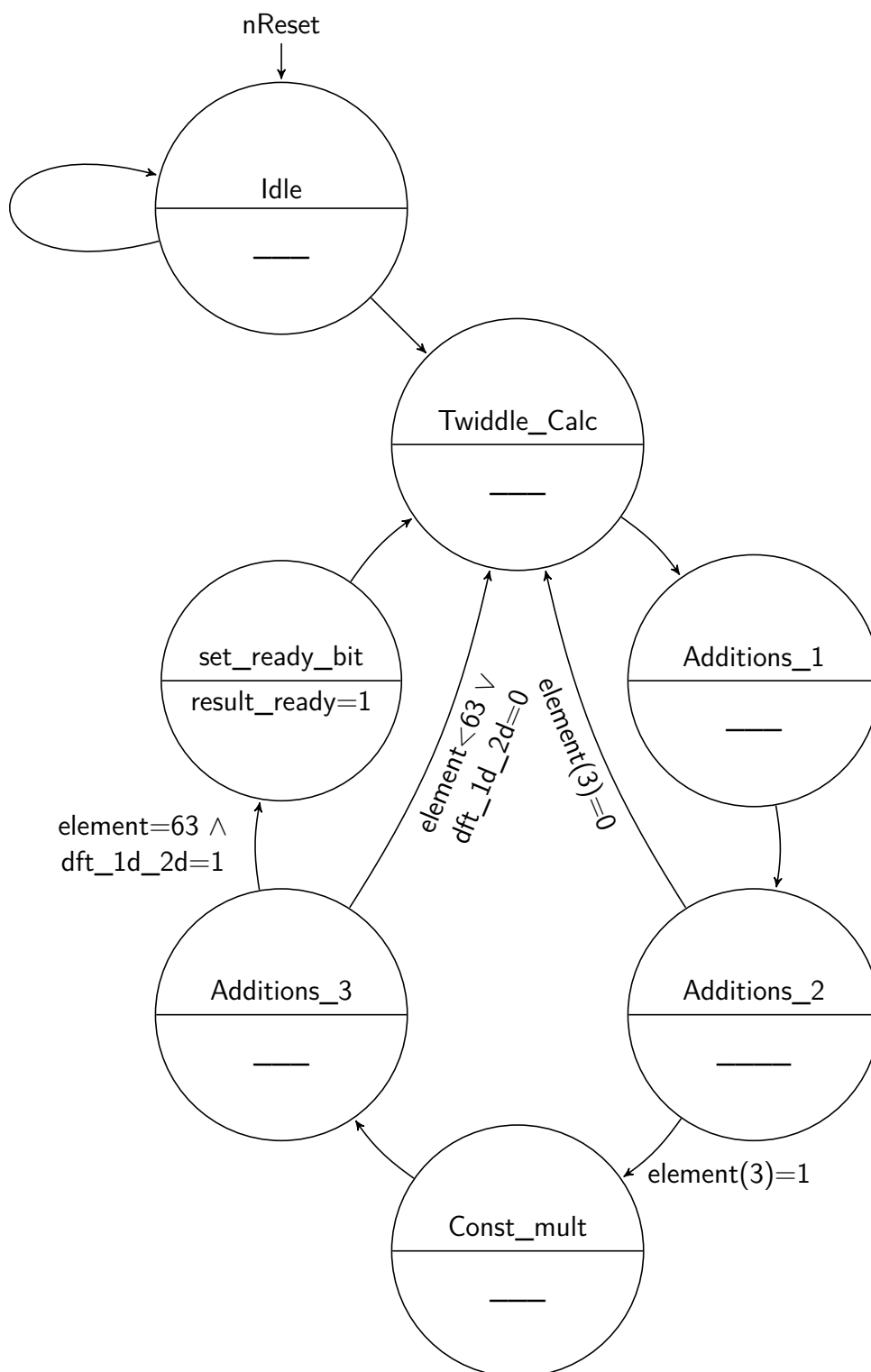
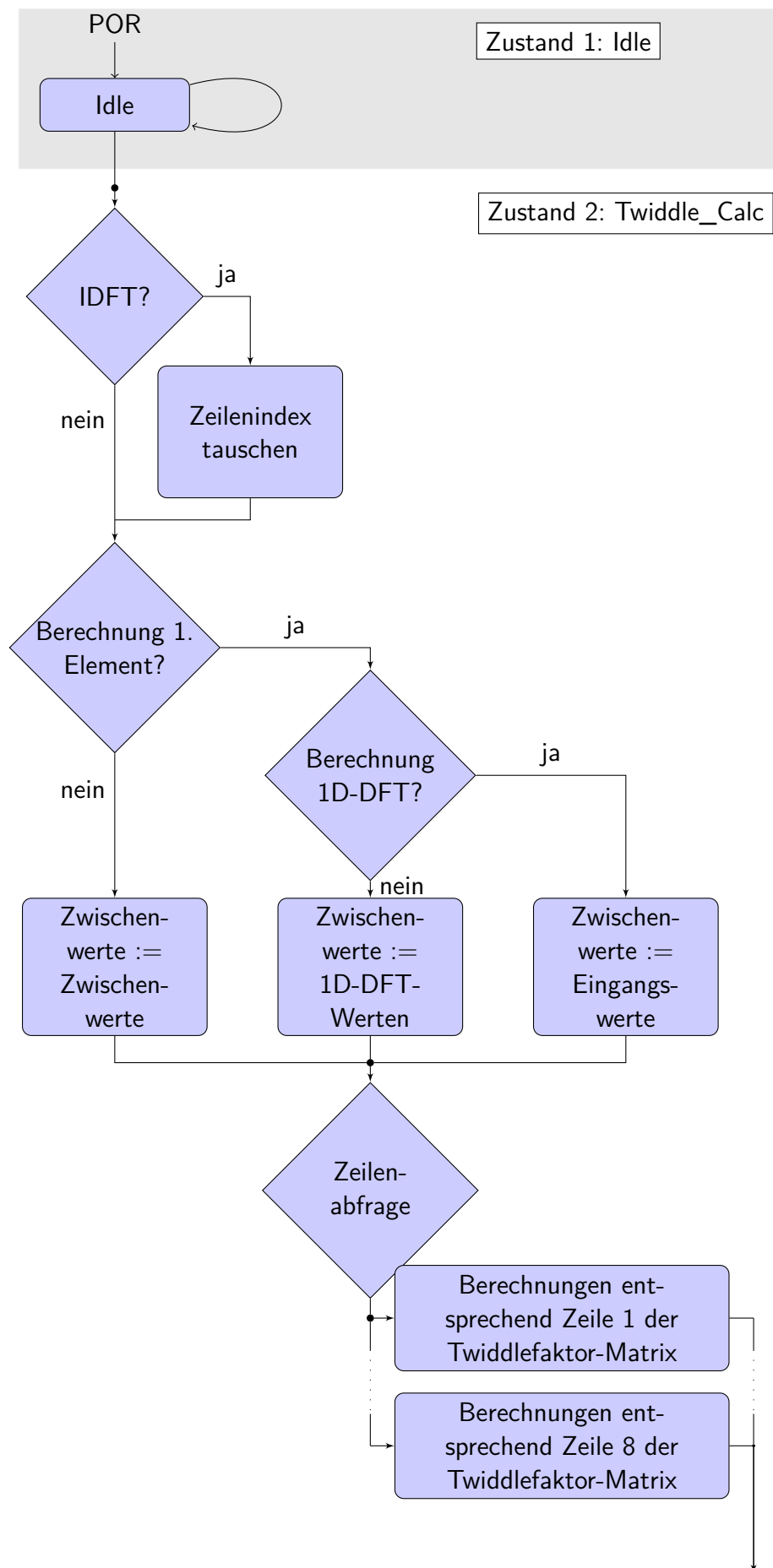
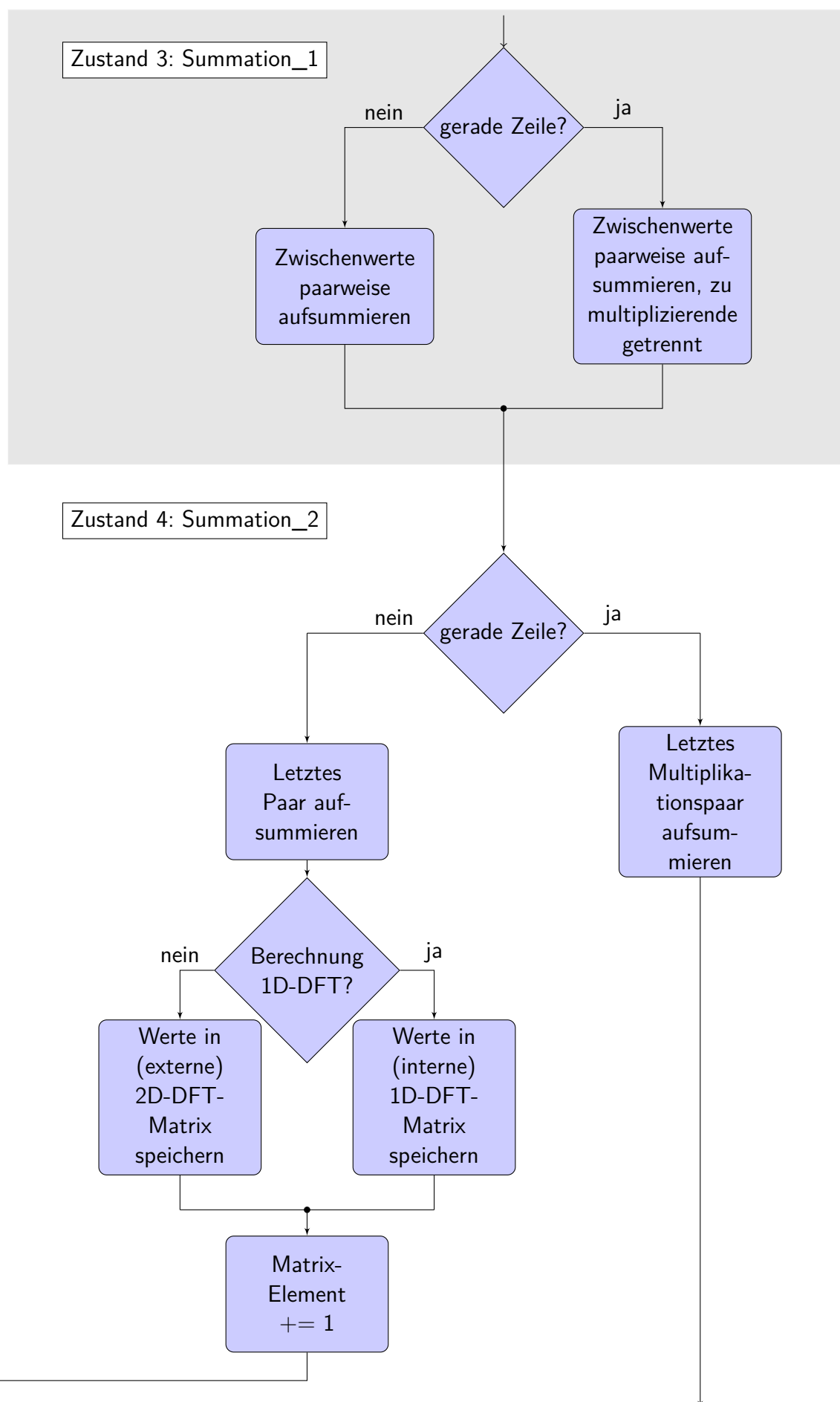
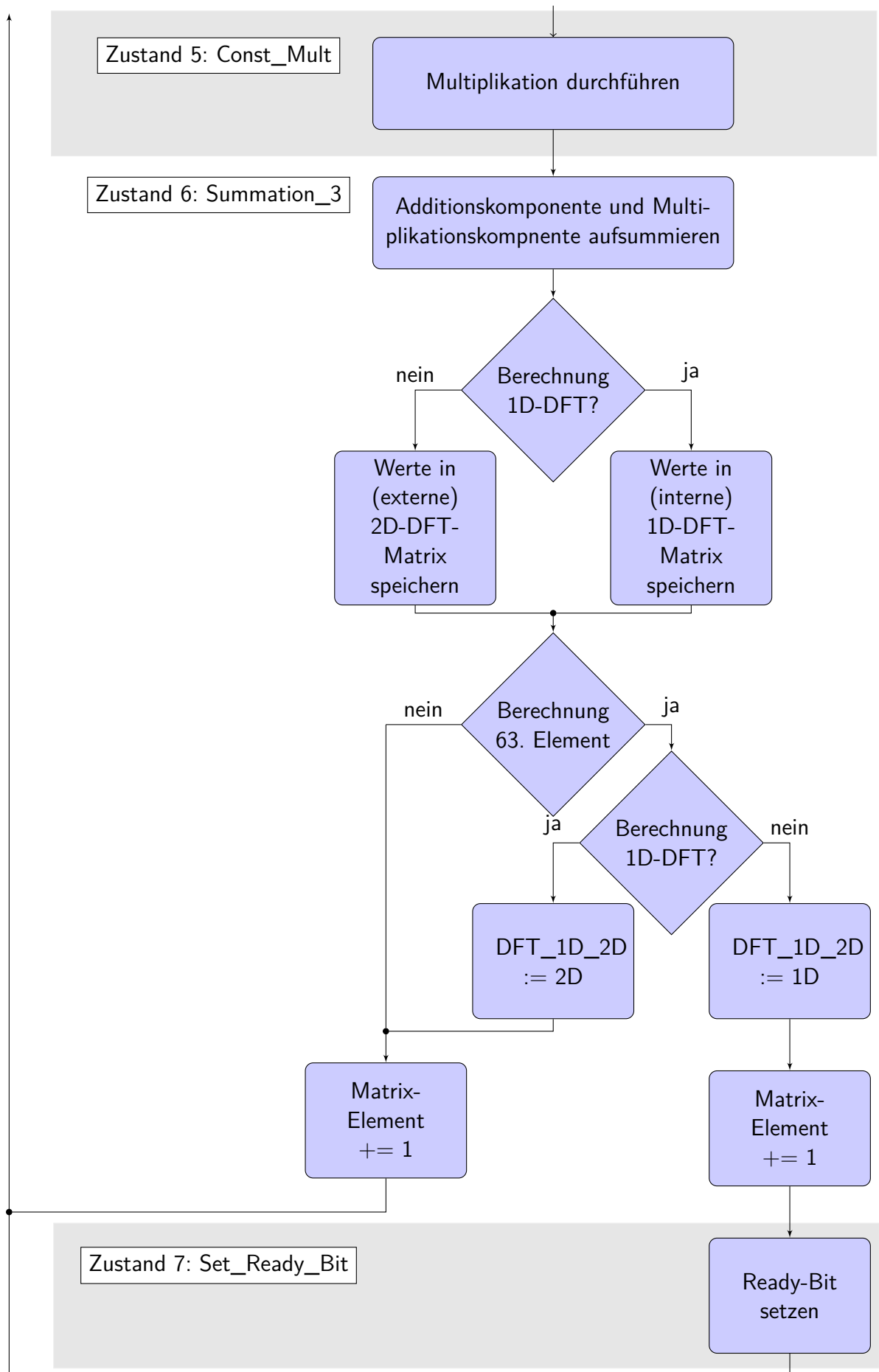


Abbildung 4.8.: Zustandsfolge der Berechnung der 1D- bzw. 2D-DFT.

4.5. UML-Diagramm







5. Evaluation

Nachdem die Implementation der 2D-DFT in VHDL erfolgt ist, werden anhand von Simulationen und eines selbstgeschriebenen Skripts Testdurchläufe durchgeführt, die die korrekte Berechnung überprüfen. Daran schließt sich die Chipimplementation an, bei der die Gatter in Form von Standardzellen des Auftragsfertigers AMS auf dem Chip platziert werden.

5.1. Simulation der 2D-DFT

Zur Simulation von Hardwarekomponenten dient in der Cadence-Umgebung das Programm NC Sim. Damit die Signalverläufe grafisch dargestellt werden, wird aus NC Sim heraus Sim-Vision gestartet. Es ermöglicht das Betrachten der Signalzustände, die im Top-Modul als externes Signal in der Entity (Eingang oder Ausgang einer Komponente) sowie internes Signal (`signal`) deklariert wurden. Wenn in der Top Level Entity instanziierte Komponenten untereinander verbunden sein sollen, müssen entsprechende Signale deklariert und ihnen zugewiesen werden. Diese Signale sind deshalb in der Simulation auch zu sehen. Signale die innerhalb einer eingebetteten Komponente Verwendung finden, sind nicht sichtbar und müssen gegebenenfalls zumindest temporär zusätzlich als Ausgang der Komponente hinzugefügt werden. Das Programm beherrscht lediglich bei einzelnen Vektoren die Umrechnung zu vorzeichenbehafteten Zahlen im Dezimalsystem. Bei der Bündelung von Vektoren können nur positive Dezimalzahlen dargestellt werden. Dies hat zur Folge, dass Vektoren, die eine negative Zahl repräsentieren, bei Bedarf händisch umgerechnet werden müssen. Erfolgen kann das mittels $2^{n+1} + m$, n = Bitbreite des Vektors und m = angezeigte Zahl. Die Darstellung von Festkommazahlen ist in keinem Fall möglich.

Anhand der Simulation kann die Anzahl der im Voraus ermittelten, zur Berechnung der 2D-DFT benötigten Takte, verifiziert werden, was nachfolgend geschehen soll.

Nachdem `nReset` auf '1' gesetzt wird, werden die Eingangswerte eingelesen. Wenn dieser Vorgang abgeschlossen ist, geht `loaded` auf '1'. Mit der nächsten steigenden Taktflanke, in Bild 5.1 bei 340 ns, beginnt die Berechnung der 2D-DFT. Beendet wird sie, nachdem die Matrizenmultiplikation auf die Eingangswerte und anschließend auf die 1D-DFT-Werte angewandt wurde. Also nach $2 \cdot 64$ einzelnen Berechnungen. Wenn dies erfolgt ist, wird `result_ready` auf '1' gesetzt. Dies geschieht bei 20 820 ns. Bei einer Taktfrequenz von $(40 \text{ ns})^{-1} = 25 \text{ MHz}$ (siehe C.8) ergeben sich so 512 Takte. Dies bestätigt auch der Edge Count, ebenfalls auf dem Bild zu sehen, welcher die Flanken des `clk`-Signals zählt. In der Simulation ist zu erkennen, dass die Berechnung der Elemente unterschiedlich viele Takte beansprucht. Hieran lässt sich ebenfalls sehen, dass die 1. (ungerade) Zeile weniger Takte gegenüber der 2. (geraden) Zeile benötigt.

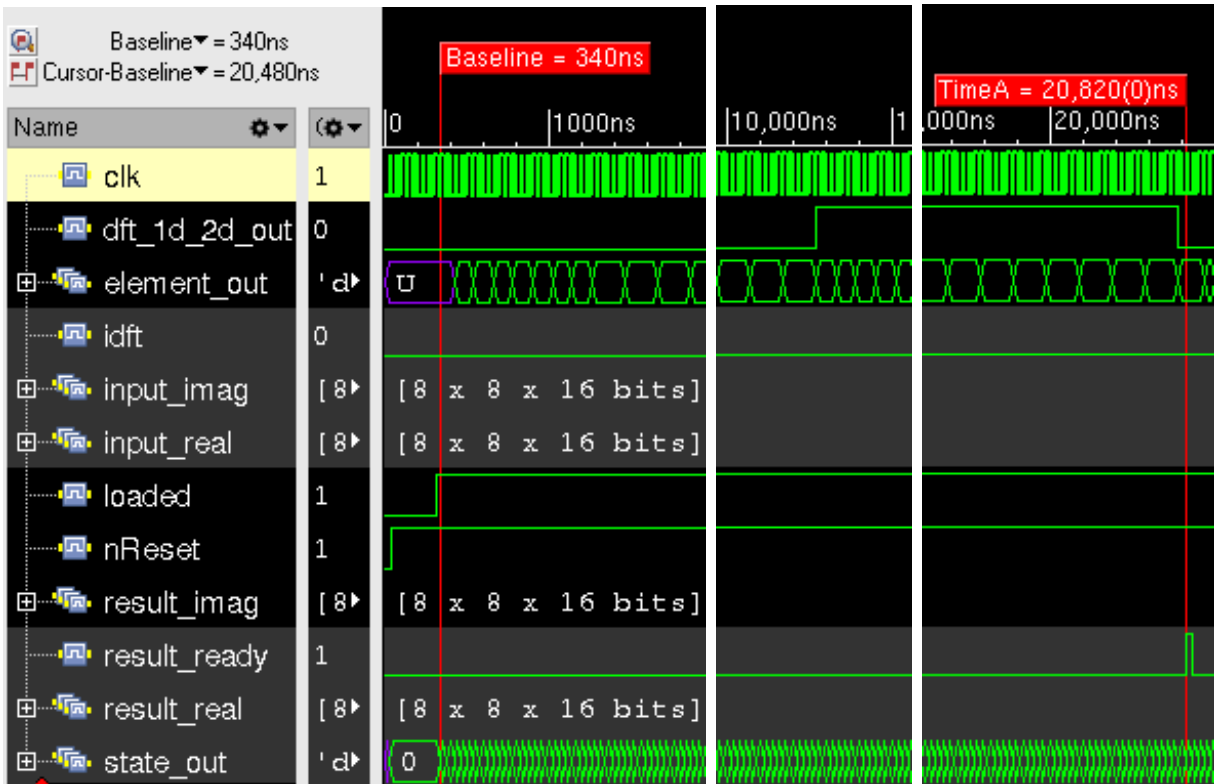


Abbildung 5.1.: Ausschnitt des Simulationstools NC Sim von der Berechnung und Verifikation der 2D-DFT.

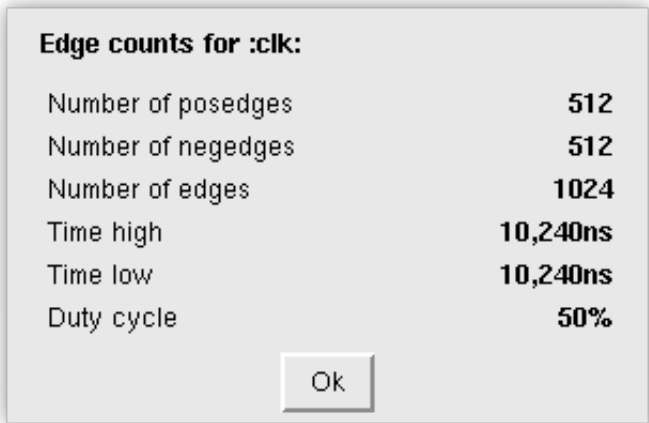


Abbildung 5.2.: Edge Count des Taktsignals für die vollständige Simulation der 2D-DFT.

5.2. Test des VHDL-Implementation der 2D-DFT

Zuerst wurde die Matrixmultiplikation mit Faktoren entworfen und getestet, die sich leicht nachvollziehen lassen. Anschließend wurden die Faktoren mit den korrekten Werten ersetzt, sodass die berechneten Werte denen der 1D-DFT entsprechen. Zuletzt wurde das Programm um die Funktionalität einer transponiert wiederholten DFT ergänzt und getestet.

5.2.1. Matrixmultiplikation mit ganzzahligen Faktoren, 1D-DFT und 2D-DFT

Der Faktor $\frac{\sqrt{2}}{2}_{10}$ ist im Quelltext als 13 Bit Vektor im S2Q10-Format abgelegt und hat den Wert 0001011010100₂. Das Waveform-Programm SimVision interpretiert diesen Wert als Ganzzahl. In der dezimalen Darstellung ist dies 724. Um die Simulationsergebnisse nachvollziehen zu können, wurden im ersten Schritt die Werte $\frac{\sqrt{2}}{2}$ der Twiddlefaktormatrix mit dem Wert $2,929\,687\,5 \cdot 10^{-3}$ ersetzt. Dieser Wert entspricht im S2Q10-Format 0000000000011 und als dezimale Ganzzahl 3. Hiermit lassen sich gut die in SimVision angezeigten Ergebnisse überprüfen. Ein weiterer Vorteil ist, dass bei diesem Faktor und den gewählten Eingangswerten der Wertebereich der 12 Bit Vektoren der Ausgangsmatrix ausreichen und es nicht zu Überläufen kommt. Aus diesem Grund konnte zu diesem Zeitpunkt auf Bitshifts nach den Additionen und Multiplikationen verzichtet werden. Erst als das binäre Äquivalent des Faktors $\frac{\sqrt{2}}{2}$ bzw. als Ganzzahl interpretiert 724 eingeführt wurde, hatten die Ergebnisse der Matrixmultiplikation mehr als die zwölf Stellen der Vektoren der Ausgangsmatrix. Mit der Änderung der Faktoren wurde es deshalb erforderlich durch Bitshifts zu verhindern, dass die Werte größer als $2^{12} = 4096$ werden. Wann und wie viele Bitshifts erfolgen geht aus den beiden Abbildungen 4.1 und 4.2 hervor.

Nachdem der Entwurf der 1D-DFT abgeschlossen war und die richtigen Ergebnisse berechnet wurden, wurde die Funktionalität der 2D-DFT ergänzt. Durch kleine Änderungen ist es leicht möglich den Programmablauf nach der Berechnung der 1D-DFT zu beenden. Um zu testen, ob Änderungen am Quelltext, welche zu fehlerhafte Berechnungen der 2D-DFT führten, auch Einfluss auf die 1D-DFT haben, war dies sehr hilfreich.

Zum Test der IDFT muss das entsprechende Eingangssignal gesetzt sein. Nun werden die Zeilen der Twiddlefaktormatrix wie in Abschnitt 4.2.5 beschrieben in umgekehrter Reihenfolge durchlaufen. Auf diese Weise lässt sich auf den mit der 2D-DFT errechneten Werten die Rücktransformation anwenden und abschließend mit den ursprünglichen Werten vergleichen.

5.2.2. Automatisierter Testdurchlauf

Häufig ist nicht der Verlauf aller Signale von Interesse ist, sondern ausschließlich die der Ergebnismatrix. Diese lassen sich ebenfalls in SimVision betrachten. Da dies nicht komfortabel ist, wurde von vornherein geplant, die Ergebnisse zusätzlich in die Datei Results.txt zu schreiben. In dieser Datei werden die logischen Pegel High und Low durch eine 1 respektive eine 0 repräsentiert. Diese Daten lassen sich beispielsweise mit dem Programm Matlab einlesen und mit der Funktion `fi()` in das gewünschte Dezimalformat konvertieren. Um die in

Hardware erfolgten Bitshifts zu kompensieren, werden die Werte mit 256 multipliziert. Die so erhaltenen Werte lassen sich nun mit der auf die Eingangswerte angewandte 1D- bzw. 2D-DFT vergleichen. Dazu müssen diese auf gleiche Weise eingelesen werden wie die Ergebniswerte.

Damit nicht zwangsläufig der gesamte Test durchlaufen werden muss, sondern sich beispielsweise auf das Generieren neuer Ergebniswerte beschränkt werden kann, erfolgt der Aufruf der VHDL Simulation und die Verifikation mit Matlab in getrennten Dateien. Die Namen der Skripte lauten `simulate.sh` und `VHDL_DFT_Vergleich.m`. Mit dem dritten Shell-Skript `check.sh` können beide aufeinanderfolgend ausgeführt werden. Der beschriebene Ablauf wird nachfolgend detaillierter und anschaulicher erläutert.

In Matlab kann die Twiddlefaktormatrix mit

$$W = e^{-\frac{i2\pi}{N} \cdot [0:N-1]' \cdot [0:N-1]}$$

berechnet werden, wobei N die Anzahl der Elemente je Zeile bzw. Spalte ist.

Bash - check.sh

```
# Dieses Skript ruft das Bash-Skript simulate.sh auf, mit dem neue Ergebnisdateien  
# erzeugt werden. Anschließend wird Matlab im Shell-Modus gestartet und das Skript  
# VHDL-DFT_Vergleich.m ausgeführt.
```

Bash - simulate.sh

```
# Dieses Skript automatisiert die Simulation der 2D-DFT in VHDL.  
# ncsim muss ein .tcl-Skript übergeben werden, in dem die Simulationsdauer  
# angegeben ist.
```

- kompilieren (alles)
- elaborieren (Top)
- simulieren (Top, alles) - -f simulationTime.tcl als Option übergeben
 - Daten einlesen (InputMatrix_komplex.txt)
 - Berechnen der 2D-DFT
 - Ergebnisse in Datei schreiben (Results.txt)

Matlab - VHDL-DFT_Vergleich.m

```
% Dieses Skript vergleicht die Berechnung der 2D-DFT in VHDL mit der DFT  
% nach Gl. (2.16).
```

- Daten einlesen (InputMatrix_komplex.txt)
- Daten von S1Q10 nach Dezimal konvertieren
- Berechnen der 2D-DFT
- Daten einlesen (Results.txt)
- Daten von S1Q10 nach Dezimal konvertieren
- Multiplikation mit 256 zur Kompensation des 8-fachen Bitshift
- Differenz beider Ergebnisse ausgeben

5.2.3. Test der 2D-DFT mit realen Eingangswerten

Für den Test realer Eingangswerte wurden die im .mat-Format vorliegenden Messwerte aus dem Verzeichnis `messung_7_7_25mm_1_31.05.2017` verwendet. Der Aufruf erfolgt wie in Abschnitt 5.2.2 beschrieben über das Skript `check.sh` in der Shell.

```
Results_vhdl_dec_real =

22.5000 -2.5000 -5.2500 -3.5000 -1.2500 -4.0000 -5.7500 -0.7500
-2.2500  2.0000  5.2500 -0.5000 -0.7500 -0.5000 -6.2500 -20.0000
 0.2500 -4.2500 -2.7500  0.7500 -1.0000         0  1.5000 -6.5000
-4.0000  0.7500 -1.2500 -2.0000 -0.7500 -1.5000         0  0.7500
-2.0000 -0.5000  1.2500         0 -0.7500 -0.7500 -2.5000 -1.2500
-3.2500 -0.5000 -1.7500  1.0000  1.0000  1.7500  0.7500 -2.2500
-2.2500  0.7500 -3.2500 -3.0000 -1.5000 -1.2500  3.2500  2.5000
-3.5000  3.0000  1.5000  0.2500 -2.0000 -2.7500 -4.2500  5.2500

Result_octave_real =

22.6982 -2.2052 -5.1230 -3.1886 -1.0303 -3.6678 -5.4980 -0.6102
-1.8929  2.0962  5.4360 -0.3486 -0.5356 -0.3005 -6.0528 -19.8764
 0.5615 -4.1421 -2.6660  0.8132 -0.7861  0.2124  1.6719 -6.3835
-3.6116  0.8337 -1.0429 -1.7275 -0.5406 -1.4101  0.0227  0.9040
-1.8506 -0.3760  1.3457  0.2868 -0.4658 -0.6260 -2.3887 -0.9879
-2.9977 -0.2522 -1.6821  1.2553  1.0747  1.8628  0.8692 -2.0928
-2.1436  0.8307 -3.0879 -2.6987 -1.4365 -1.1042  3.3398  2.8628
-3.1227  3.2538  1.6171  0.5159 -1.7172 -2.6618 -4.0266  5.5576

diff_real =

0.1982  0.2948  0.1270  0.3114  0.2197  0.3322  0.2520  0.1398
0.3571  0.0962  0.1860  0.1514  0.2144  0.1995  0.1972  0.1236
0.3115  0.1079  0.0840  0.0632  0.2139  0.2124  0.1719  0.1165
0.3884  0.0837  0.2071  0.2725  0.2094  0.0899  0.0227  0.1540
0.1494  0.1240  0.0957  0.2868  0.2842  0.1240  0.1113  0.2621
0.2523  0.2478  0.0679  0.2553  0.0747  0.1128  0.1192  0.1572
0.1064  0.0807  0.1621  0.3013  0.0635  0.1458  0.0898  0.3628
0.3773  0.2538  0.1171  0.2659  0.2828  0.0882  0.2234  0.3076
```

Listing 5.1: Test der VHDL-Implementation der 2D-DFT mit Messwerten, die am Sensor-Messplatz aufgenommen wurden.

In der ersten der drei Matrizen stehen die in VHDL berechneten Werte, konvertiert und skaliert, damit sie mit den in Matlab (Octave) aus der zweiten Matrix vergleichbar sind. In der dritten Matrix sind die Differenzen der jeweiligen Werte zu sehen, welche zwischen knapp 0,1 und knapp 0,4 liegen. Auf die Darstellung der Eingangswerte sowie der prozentualen Differenz wurde aus Gründen der Übersicht verzichtet. Gezeigt werden kann dennoch, dass trotz der insgesamt acht Bitshifts je Zahl die Abweichung zwar relativ groß aber vermutlich noch in einen annehmbaren Rahmen liegt.

5.3. Zeitabschätzung als Winkelsensor im Antriebsmotor eines Elektroautos

Anhand der aus Abbildung 5.1 bekannten Größe von 512 Takten für die 2D-DFT kann zusammen mit den gegebenen Informationen ermittelt werden, ob die Implementation vom zeitlichen Aspekt her akzeptabel ist. Die maximale Drehzahl eines Elektromotors wird im Datenblatt der Firma ABM Greiffenberger mit 8000 U/min angegeben [11, S. 5], als maximale Taktfrequenz des ASIC ist 100 MHz vorgesehen und es wird eine Winkelgenauigkeit von 1° angestrebt. Aus diesen Angaben lässt sich errechnen, ob die Vorgaben eingehalten werden und wie viele Takte für andere Berechnungen zur Verfügung stehen.

$$RPM = 8000 \text{ min}^{-1}$$

$$\frac{RPM}{60} = 133,3 \text{ sec}^{-1} \quad (5.1)$$

$$\begin{aligned} \curvearrowright 1 \text{ Umdrehung} &= \frac{1}{133,3 \text{ sec}^{-1}} \\ &= 7,5 \cdot 10^{-3} \text{ sec} \end{aligned} \quad (5.2)$$

$$\begin{aligned} 1^\circ &\hat{=} \frac{7,5 \cdot 10^{-3} \text{ sec}}{360} \\ 1^\circ &\hat{=} 20,83 \cdot 10^{-6} \text{ sec} \end{aligned} \quad (5.3)$$

$$100 \cdot 10^6 \text{ Hz} = 10 \cdot 10^{-9} \text{ sec}$$

$$\frac{20,83 \cdot 10^{-6} \text{ sec}}{10 \cdot 10^{-9} \text{ sec}} = 2083 \text{ Takte} \quad (5.4)$$

Um eine Aussage über zur Verfügung stehende Zeit bzw. Takte machen zu können, wurde in Gleichung (5.4) gezeigt, dass für je 1° Rotation des Motors etwa 2080 Takte für Berechnungen zu Verfügung stehen. Die 512 für die 2D-DFT sind etwa 25% von 2080. Daraus resultiert, dass noch etwa 75% der verfügbaren Takte nutzbar sind. Wenn auch die IDFT durchgeführt werden muss, bleibt zumindest die Hälfte der Takte für Interpolation, Filterung, Berechnung sowie Ausgabae bzw. Übertragung der gewonnenen Informationen.

5.4. Chipdesign

Um die Fertigung eines ASIC in Auftrag geben zu können, sind nach der Implementation in einer Hardwarebeschreibungssprache wie VHDL noch weitere Schritte notwendig. Der erste davon ist die Simulation des kompilierten Quelltextes ohne weitere Anpassungen an die Stan-

dardzellen oder den verwendeten Prozess zur Verifikation, ob alle Komponenten für sich und auch das Gesamtsystem, wie erwartet funktionieren. Darauf folgt eine Timingsimulation, in der für die Standardzellen spezifische Verzögerungszeiten des Herstellers eingebunden werden, welche gegebenenfalls vorhandene symbolische Laufzeiten für die Logikgatter mit realistischeren Werten ersetzen.

5.4.1. Syntheseergebnis der 2D-DFT-Einheit

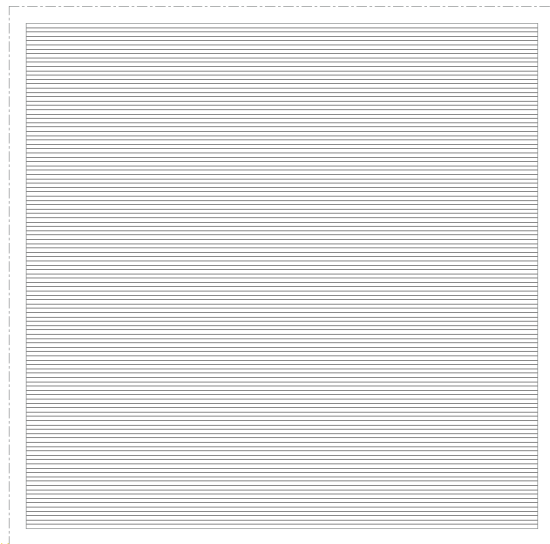
In Abbildung 5.3 ist die mit dem Programm RTL Compiler (rc) erzeugte Netzliste als Syntheseergebnis der 2D-DFT des VHDL-Codes zu sehen. Auch wenn keine einzelnen Gatter erkennbar sind, wird deutlich, dass auf die Verdrahtung ein sehr hoher Anteil der Fläche entfällt. In den bisherigen Darstellungen wurden die Leitungen schwarz und die Standardzellen rot eingefärbt, hierauf wurde in diesem Fall bewusst verzichtet. Die Anzahl der benötigten Standardzellen liegt bei 15 310. Ihr Flächenbedarf wird von dem Synthesewerkzeug mit $1\,524\,960\,\mu\text{m}^2$ angegeben, also etwa $1,5\text{ mm}^2$. Darin ist der Bedarf für die Leitungen nicht mit eingerechnet. Ersichtlich ist, dass dieser deutlich höher liegt. Das Synthesewerkzeug rc platziert alle Standardzellen auf einer Ebene, für Verbindungen werden zwei Lagen genutzt. Beim Floorplan werden die Standardzellen und Leitungen in unserem Fall auf drei und allgemein mit bis zu über zehn Lagen verteilt, weshalb eine deutlich kleinere Fläche ausreicht. Wie viele Lagen genutzt werden können hängt von der verwendeten Technologie und somit Strukturgröße zusammen.



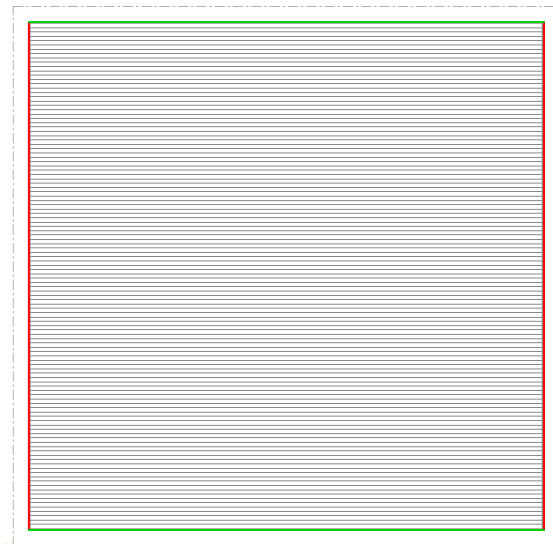
Abbildung 5.3.: Mit RTL compiler (rc) erstellte Netzliste der 2D-DFT-Einheit.

5.4.2. Floorplan der 2D-DFT-Einheit mit 3 Metalllagen und 350nm Strukturgröße

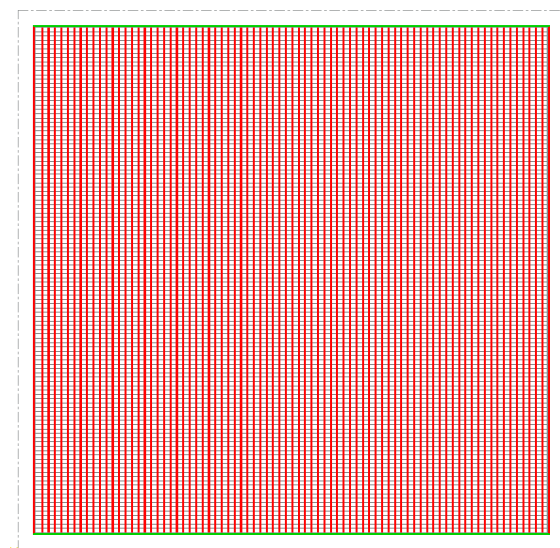
In den folgenden Bildern 5.4a bis 5.4d ist zusehen, wie der Floorplan eine Stromversorgung für die Standardzellen erhält. In Abbildung 5.4a ist noch keinerlei Stromversorgung vorhanden. Als erstes wird, wie in Abb. 5.4b zu sehen, der äußere Ring gesetzt, welcher alle weiteren Verbindungen mit Strom versorgt. Die unterschiedlichen Farben stehen für verschiedene Ebenen des Chips. Bei rot handelt es sich um die zweite, bei grün um die dritte Lage. Besser zu sehen ist die in Abbildung 5.5. Dort lassen sich auch die zur Durchkontaktierung der einzelnen Lagen genutzten Vias erkennen (rot eingekreist). Desweiteren lassen sich Masse (innerer Ring) und Spannungsversorgung (V_{dd} , äußerer Ring) unterscheiden. Ersichtlich ist darüber hinaus, dass für die horizontalen und vertikalen Leitungen unterschiedliche Lagen genutzt werden. Auf diese Weise wird vermieden, dass es zu ungewollten Verbindungen von Leitungen kommt. In Abbildung 5.4c werden, wieder auf Layer zwei, vertikalen Leitungen zur Stromversorgung der Standardzellen hinzugefügt. Auf Bild 5.4d werden abschließend auf Layer eins horizontale Verbindungen ergänzt. Auf diese Weise ist die Voraussetzung geschaffen, dass aktive Standardzellen mit Strom versorgt werden können und die Netzliste aus Abbildung 5.3 auf den Floorplan übertragen werden kann. Das Resultat davon ist in Abbildung 5.6 zu sehen.



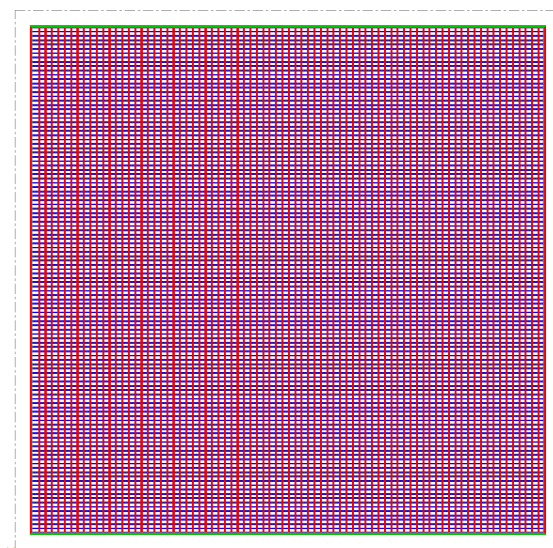
(a) Leerer Floorplan



(b) Floorplan mit Power-Ring



(c) Floorplan mit vertikaler Stromversorgung auf Metal-Layer 2



(d) Floorplan mit zusätzlicher horizontaler Stromversorgung auf Metal-Layer 1

Abbildung 5.4.: Schrittweiser Entwurf der Stromversorgung des Floorplans

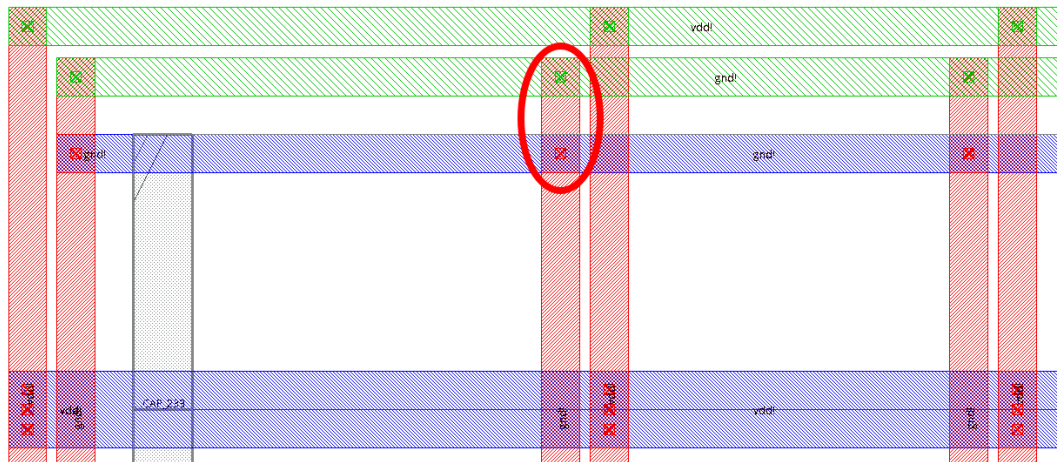


Abbildung 5.5.: Vias für die Durchkontaktierung der Stromversorgung auf den verschiedenen Metalllagen (blau: Layer 1, rot: Layer 2, grün: Layer 3).

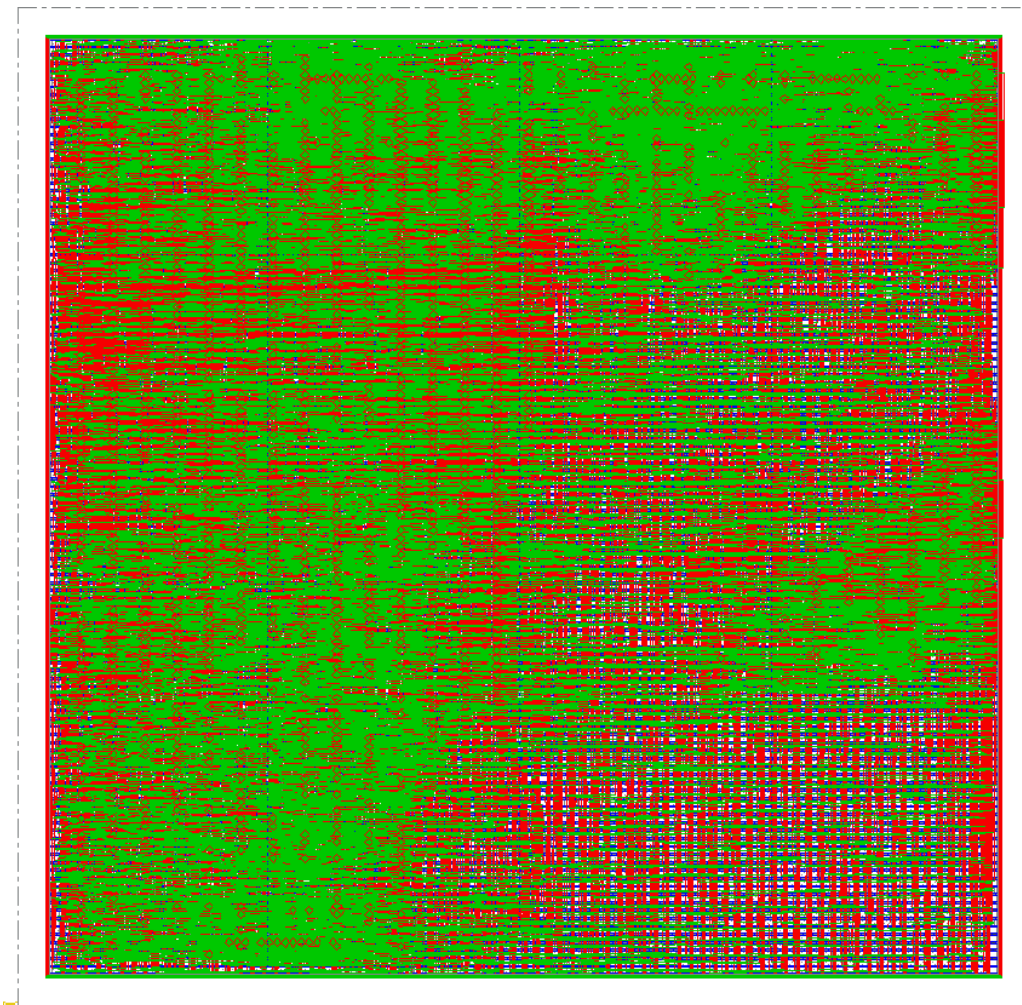


Abbildung 5.6.: Floorplan mit platzierten Standardzellen der 2D-DFT-Einheit.

6. Schlussfolgerungen

Abschließend werden die wichtigsten Aspekte und Erkenntnisse dieser Arbeit zusammengetragen und bewertet. Im Ausblick werden zwei Kritikpunkte aufgegriffen und Lösungsansätze aufgezeigt.

6.1. Zusammenfassung

Diese Arbeit ist Teil des ISAR-Projekts der HAW-Hamburg, welches einen ASIC entwickelt, auf dem Signalgewinnung mittels magnetoresistiver Sensoren und deren Verarbeitung vereint werden. Es wird eine 2D-DFT in VHDL implementiert, welche die Eingangssignale eines Sensor-Arrays zur Filterung in den Bildbereich transformiert.

Damit eine effiziente Nutzung der Chipfläche und der zur Verfügung stehenden Takte möglich sind, werden im ersten Schritt relevante Größen der DFT und der DCT bezüglich Anzahl nicht trivialer Faktoren der Twiddlefaktormatrix einander gegenübergestellt. Die Entscheidung fällt auf die 8x8-DFT, da ihre Twiddlefaktormatrix nur einen nicht trivialen Faktoren hat. Real- und Imaginärteil sind darüber hinaus betragsmäßig identisch. Dieser tritt nur in der Hälfte der Zeilen auf, weshalb die Anzahl vollwertiger Multiplikationen für die 1D-DFT auf 64 reduziert werden kann. Dieser Wert ist identisch mit dem der benötigten Multiplikationen bei der Berechnung mittels FFT-Algorithmus. Tauschen der Spalten- und Zeilenindizes der 1D-DFT-Matrix transponiert diese, was es ermöglicht, die 2D-DFT ohne Umspeichern zu berechnen. Durch Tauschen der Zeilen der Twiddlefaktormatrix ist es auf einfache Weise möglich, die IDFT zu realisieren.

Eine Implementierung als FFT wird nicht in betracht gezogen, da sich die gewonnen Erkenntnisse nicht auf Transformationen mit einer ungeraden Anzahl Elemente übertragen ließen. In einer auf diese Bachelorarbeit aufbauenden Masterarbeit wird eine 2D-DFT der Größe 15x15 entwickelt. Ein Ansatz für eine schnellere Berechnung wäre die reelle Matrixmultiplikation. Bei ihr würden Eingangssignal und das Ergebnis der 1D-DFT in Real- und Imaginärteil aufgeteilt und separat transformiert werden. Wesentlicher Nachteil dieser Herangehensweise wäre, dass mehr Speicher und Leitungen benötigt würden.

Abschließend wird die Funktionalität des VHDL-Quelltextes anhand einer Simulation verifiziert und der Floorplan als wesentlicher Schritt für die Chipimplementation durchgeführt.

6.2. Bewertung und Fazit

Es wurde eine schnelle Berechnung der 2D-DFT implementiert, die mit der Effizienz der FFT vergleichbar ist, sich aber im Ansatz unterscheidet und besser auf andere Größen einer DFT übertragen lässt. Mit dieser Aussage geht jedoch nicht einher, dass sich andere Größen genauso

effizient implementieren lassen. Die wesentliche Erkenntnis ist, dass die Anzahl verschiedener nicht trivialer Faktoren ausschlaggebend für den Aufwand der Implementierung ist.

Für die Berechnung der 1D-DFT kann auch die Einheit für die Berechnung der 2D-DFT verwendet, und vollständig auszunutzt werden, wenn die Eingangssignale, welche aus einer Sinus- und einer Kosinuskomponente bestehen, zu einem komplexen Signal zusammengefasst werden.

Voraussetzung für die effiziente Implementierung ist, dass die Twiddlefaktormatrix identisch mit ihrer Transponierten ist und die Ergebnismatrix durch vertauschen der Spalten- und Zeilenindizes erfolgen kann. Die DFT hat unter diesen Aspekten deutliche Vorteile gegenüber der DCT.

Die DFT ist trotz ihrer komplexen Twiddlefaktoren besser als die DCT für die Transformation geeignet ist. Begründet wird dies mit der Anzahl nicht trivialer Faktoren und mit der unterschiedlichen Symmetrie, die die beiden Arten von Twiddlefaktormatrizen aufweisen. Die der DFT lässt sich in vier Viertel, die der DCT in zwei Hälften aufteilen, weswegen letztere nicht identisch mit ihrer Transponierten sein kann. Dies wiederum erschwert es, die Berechnung der 2D-DFT mit der Einheit für die Berechnung der 1D-DFT durchzuführen.

Es wird angenommen, dass für die gesamte Signalverarbeitung etwas 2800 Takte zur Verfügung stehen. Für die 8x8-DFT werden 512 Takte benötigt, was 25% der vorhandenen Takte entspricht. Wenn auch die IDFT berechnet werden soll, bleiben noch etwa die Hälfte aller Takte für die restliche Signalverarbeitung. Es kann davon ausgegangen werden, dass weniger Takte notwendig sein werden. Im Umkehrschluss bedeutet dies, dass bei gleicher Anforderung die 15x15-DFT nur unwesentlich mehr Takte in Anspruch nehmen darf. Dies kann neben dem gewonnen Wissen als wichtigster Richtwert an die anküpfende Masterarbeit weitergegeben werden.

Für die nicht trivialen Faktoren ist eine Konstantenmultiplikation erforderlich, welche über ein Schaltnetz realisiert werden kann. Hierfür sind bei gleicher Bitbreite weniger als halb so viele Standardzellen erforderlich. Dies bedeutet eine deutliche Reduzierung des benötigten Platzes. Diese Angabe trifft jedoch keine Aussage über die Anzahl von hintereinander geschalteten Gattern, welche um etwa 25% höher als die Bitbreite ist. Dies ist ein entscheidender Wert zur Ermittlung der maximalen Taktfrequenz. Darüber hinaus ist es bei anderen Faktoren möglich, dass mehr verschachtelte Gatter erforderlich sind. Dieser Wert hängt mit der Anzahl der Einsen zusammen, die notwendig sind, um eine Zahl binär zu repräsentieren. Basierend auf dem in den entsprechenden Vorlesungen Gelernten, ist davon auszugehen, dass dies als ein kritischer Pfad der gesamten Schaltung angenommen werden kann.

6.3. Ausblick

Anders als angenommen, hat Cadence bezüglich der Multiplikationen keine tiefgreifende Optimierung vorgenommen. Der durch die Konstantenmultiplikation auftretende kritische Pfad kann auf verschiedene Weise optimiert werden. Die einfachste Lösung wäre es, das Schaltnetz mit dem Programm RTL Compiler zu erzeugen, in zwei getrennte Schaltnetze aufzuteilen und

die Ausgänge des ersten als Eingänge des zweiten zu nutzen. Auf diese Weise ließen sich die Gatterlaufzeiten auf zwei Takte aufteilen.

Denkbar ist ebenfalls, das Schaltnetz des Konstantenmultiplizierers zu minimieren. Ein vielversprechender Ansatz ist die Anwendung des Wallace-Tree-Verfahrens. Mit ihm können die Anzahl der Gatter, insbesondere der hintereinander geschalteten, reduziert und so die Berechnung beschleunigt werden [12, S. 8-10].

Um die numerische Ungenauigkeiten, die durch das Bitshiften auftreten, zu reduzieren, ist es sinnvoll an dieser Stelle mehr Aufwand zu investieren. Ein erster Ansatz wäre es, zu prüfen, ob der Bitshift nach der Subtraktion erforderlich ist. Da diese Arbeit den Schwerpunkt in der Aufwandsabschätzung einer Chipimplementation einer 2D-DFT auf einem ASIC hat, ist diese Problematik kein Gegenstand dieser Arbeit und wird an dieser Stelle nur in Grundzügen erwähnt.

Abkürzungsverzeichnis

1D-DFT	eindimensionale diskrete Fouriertransformation
2D-DFT	zweidimensionale diskrete Fouriertransformation
ADC	Analog Digital Converter
AMR	anisotroper magnetoresistiver Effekt
ASIC	Application Specific Integrated Circuit, <i>dt.: Anwendungsspezifischer Integrierter Schaltkreis</i>
DCT	diskrete Kosinus Transformation
DFT	diskrete Fouriertransformation
FFT	Fast Fouriertransformation
FT	Fouriertransformation
IDFT	inverse diskrete Fouriertransformation
ISAR	<i>Signalverarbeitung für integrated Sensor-Arrays basierend auf dem Tunnel-Magnetoresistiven Effekt für den Einsatz in der Automobilelektronik</i>
LSB	Least Significant Bit
MSB	Most Significant Bit
RAM	Random Access Memory
TMR	tunnelmagnetoresistiver Effekt

Abbildungsverzeichnis

1.1. Ablauf der Signalvorverarbeitung [1, S. 9]	V
2.1. Interpretation von Dualzahlen im SQ3-Format.	4
2.2. Veranschaulichung der Matrixmultiplikation.	5
2.3. Einheitskreis, Zusammensetzung des komplexen Zeigers e^{jkt} aus einer Sinus- und einer Kosinusfunktion mit dem selben Argument.	6
2.4. 1D-DFT als Matrixmultiplikation.	9
2.5. 2D-DFT als Matrixmultiplikation.	9
2.6. Veranschaulichung der Berechnung der DFT mit reellen Eingangswerten. . . .	11
2.7. Redundante Werte der spaltenweisen DFT einer 8x8-Matrix. Der Imaginärteil der redundanten Werte hat denselben Betrag mit negiertem Vorzeichen.	12
3.1. Einheitskreis mit relevanten Werten der 8x8-DFT.	16
3.2. Twiddlefaktoren der 8×8 -Matrix, aufgeteilt auf die Laufindizes m und n . m bezieht sich auf das Element im Ausgangsvektor \vec{X} , n auf den Eingangsvektor \vec{x} . Siehe auch Gl. (2.11).	17
3.3. Matrix-Darstellung der 8x8-DFT-Twiddlefaktoren aufgeteilt nach Real- und Imaginärteil.	18
3.4. Berechnungsschema der DFT mit acht Eingangswerten nach dem Butterfly-Verfahren.	22
4.1. Vorgehensweise der Akkumulation der ungeraden Spalten der Eingangswerte. .	26
4.2. Vorgehensweise der Akkumulation und Multiplikation der geraden Spalten der Eingangswerte.	27
4.3. Darstellung der Berechnung der 2D-DFT aus Gleichung (4.1).	30
4.4. Vertauschen der Zeilen 2 und 8, 3 und 7 sowie 4 und 6 der Twiddlefaktormatrix, um von der DFT zur IDFT zu gelangen.	30
4.5. Schaltnetz des 13 Bit Konstantenmultiplizierers für $\frac{\sqrt{2}}{2} = 0.70711 \simeq 0.70703125 = 0001011010100_2$ in Encounter; Eingang links, Ausgang rechts. .	32
4.6. Schaltnetz einer Einheit zur Bildung des 2er-Komplements eines 13 Bit Vektors; Eingang links, Ausgang rechts.	33
4.7. Schaltnetz eines 12 Bit Addierers, Eingänge links (12 Bit), Ausgang rechts (13 Bit).	33
4.8. Zustandsfolge der Berechnung der 1D- bzw. 2D-DFT.	35
5.1. Ausschnitt des Simulationstools NCSim von der Berechnung und Verifikation der 2D-DFT.	40
5.2. Edge Count des Taktsignals für die vollständige Simulation der 2D-DFT. . . .	40

5.3. Mit RTL compiler (rc) erstellte Netzliste der 2D-DFT-Einheit.	47
5.4. Schrittweiser Entwurf der Stromversorgung des Floorplans	49
5.5. Vias für die Durchkontaktierung der Stromversorgung auf den verschiedenen Metalllagen (blau: Layer 1, rot: Layer 2, grün: Layer 3).	50
5.6. Floorplan mit platzierten Standardzellen der 2D-DFT-Einheit.	50

Tabellenverzeichnis

2.1. Wahrheitstabelle eines XOR-Gatters. A und B sind Eingänge, X das Ausgangs- signal	3
3.1. Bewertung der DCT-Twiddlefaktor-Matrizen	15
3.2. Bewertung der DFT-Twiddlefaktor-Matrizen	16
3.3. Auflistung benötigter reeller Multiplikationen der verschiedenen Methoden für die 2D-DFT	22
4.1. Benötigte Takte für die komplexe DFT	28
4.2. Vergleich der Synthesergebnisse der Teilkomponenten in Bezug auf Anzahl der Gatter und ihre Fläche	32

Literatur

- [1] K.-R. Riemschneider und T. Schütthe, *FREQUENCY FILTERING AND STRAY FIELD COMPENSATION USING 2D-DFT ALGORITHM*.
- [2] jinfengl, *MultiDimension (MDT) TMR Magnetic Sensors*. Adresse: https://www.sensor-test.de/ausstellerbereich/upload/mnpdf/en/Flyer0429_13.pdf.
- [3] Y. Tsukakoshi, *What's the Difference Between TMR and GMR Sensors?*, Juni 2017. Adresse: <http://www.electronicdesign.com/industrial-automation/what-s-difference-between-tmr-and-gmr-sensors>.
- [4] J. Reichardt, *Lehrbuch Digitaltechnik: Eine Einführung mit VHDL*. Walter de Gruyter, 2013.
- [5] L. Papula, „Mathematik für Ingenieure und Naturwissenschaftler: Ein Lehr-und Arbeitsbuch für das Grundstudium (Band 2)“, *Vieweg+ Teubner*, Jg. 14, 2015. Adresse: <https://link.springer.com/content/pdf/10.1007%2F978-3-658-07790-7.pdf>.
- [6] K. D. Tönnis, *Grundlagen der Bildverarbeitung*, Pearson, 2005.
- [7] J. G. Proakis und D. G. Manolakis, *Digital signal processing: principles, algorithms, and applications*. Pearson Prentice Hall, 2007.
- [8] W. Burger und M. J. Burge, *Digitale Bildverarbeitung: Eine Einführung mit Java und ImageJ*, 2. Aufl. Springer, 2006. Adresse: <https://link.springer.com/content/pdf/10.1007%2F3-540-30941-1.pdf>.
- [9] P. Sturm, *Bild- und Videokompressionstechniken: Theorie und Transformationskodierung*, 1999. Adresse: <https://www.uni-koblenz.de/~lb/lehre/ws2006/bv/Sturm-1999-Theorie%20der%20Transformationskodierung.pdf>.
- [10] Wikipedia, *Wikipedia: Diskrete Kosinustransformation*, Apr. 2018. Adresse: https://de.wikipedia.org/wiki/Diskrete_Kosinustransformation.
- [11] A. Greiffenberger, *Motoren für Elektrofahrzeuge - Energieeffizient Langlebig Kompakt*. Adresse: http://www.abm-drives.com/file/5864_Motoren_f%C3%BCr_Elektrofahrzeuge.pdf.
- [12] J. Drechsler, *Binäre Multiplikations- und Divisionswerke*, Juli 2008. Adresse: https://www.ra.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_RA/tgi2/rechenwerke.pdf.

Anhang

A. Matlab-Skripte

A.1. Skript zur Bewertung von Twiddlefaktormatrizen

```
2 %% Dateiname: dct_bewertung.m
3 %% Funktion: Bewertet die Koeffizienten der DCT-Twiddlefaktormatrix
4 %%           darauf basierend, wie trivial die Berechnungen mit
5 %%           den Twiddlefaktoren sind.
6 %%           Als trivial gelten Berechnungen mit den Werten -1, -0.5, 0,
7 %%           +0.5, +1
8 %%           Es wird ein Verhaeltnis aus trivialen und nicht trivialen
9 %%           Werten
10 %%           erstellt.
11 %% Argumente: N (Groesse der NxN DCT-Matrix)
12 %% Author:    Thomas Lattmann
13 %% Datum:     17.10.2017
14 %% Version:   1.0
15
16 function dct_bewertung(N)
17
18     % Twiddlefaktor-Matrix erzeugen
19     W = cos(pi/N*(0:N-1)')*(0:N-1+.5));
20     W = round(W*1000000)/1000000;
21
22     % Werte kleiner 0,000001 auf 0 setzen (arithmetische Ungenauigkeiten)
23     W(abs(W) < 0.000001) = 0;
24
25     % Anzahl verschiedener Werte ermitteln
26     different_nums = unique(W);
27     different_non_trivial_nums = different_nums(find(different_nums ~= 1));
28     different_non_trivial_nums = different_non_trivial_nums(find(
29         different_non_trivial_nums ~= -1));
30     different_non_trivial_nums = different_non_trivial_nums(find(
31         different_non_trivial_nums ~= 0.5));
32     different_non_trivial_nums = different_non_trivial_nums(find(
33         different_non_trivial_nums ~= -0.5));
34     different_non_trivial_nums = different_non_trivial_nums(find(
35         different_non_trivial_nums ~= 0));
36
37     different_non_trivial_nums = unique(abs(different_non_trivial_nums));
38     different_non_trivial_nums
39     %non_trivial = length(abs(different_non_trivial_nums))
```

```

36 % Jeweils die Menge der verschiedenen Werte ermitteln
num_count = zeros(1, length(different_nums));
38 for k = 1:length(different_nums)
    for n = 1:N
40         for m = 1:N
            if different_nums(k) == W(m,n)
42                 num_count(k) = num_count(k) + 1;
            end
44         end
    end
46 end

48 % nicht triviale Werte der Matrix zaehlen
nontrivial_nums = 0;
50 for k = 1:length(different_nums)
    if abs(different_nums(k)) != 1
52         if abs(different_nums(k)) != 0.5
            if different_nums(k) != 0
54                 nontrivial_nums = nontrivial_nums + num_count(k);
            end
56         end
    end
58 end
60
62 nums_of_matrix = N*N;
64 trivial_nums = N*N - nontrivial_nums
66 nontrivial_nums
68 v = trivial_nums/nontrivial_nums
end

```

Listing A.1: Octave-Skript zur Bewertung unterschiedlicher DCT-Twiddlefaktormatrizen

```

1 %% Dateiname: dft_bewertung.m
2 %% Funktion: Bewertet die Koeffizienten der DFT-Twiddlefaktormatrix
3 %%           darauf basierend, wie trivial die Berechnungen mit
4 %%           den Twiddlefaktoren sind.
5 %%           Als trivial gelten Berechnungen mit den Werten -1, -0.5, 0,
6 %%           +0.5, +1
7 %%           Es wird ein Verhaeltnis aus trivialen und nicht trivialen
8 %%           Werten
9 %%           erstellt.
10 %% Argumente: N (Groesse der NxN DFT-Matrix)
11 %% Author: Thomas Lattmann
12 %% Datum: 17.10.2017
13 %% Version: 1.0

14 function dft_bewertung(N)

```

```

15 % Twiddlefaktor-Matrix erzeugen
W = exp(-i*2*pi*[0:N-1]'.*[0:N-1]/N);
17 W = round(W*1000000)/1000000;

19 % Matrix nach Im und Re trennen und Werte runden
W_r = real(W);
21 W_i = imag(W);

23 % Werte kleiner 0,000001 auf 0 setzen (arithmetische Ungenauigkeiten)
W_r(abs(W_r) < 0.000001) = 0;
25 W_i(abs(W_i) < 0.000001) = 0;

27

29 % Anzahl verschiedener Werte ermitteln
different_nums_real = unique(W_r);
31 different_nums_imag = unique(W_i);

33 different_nums = [different_nums_real; different_nums_imag];
different_nums = unique(different_nums);
35 different_non_trivial_nums = different_nums(find(different_nums ~= 1));
different_non_trivial_nums = different_non_trivial_nums(find(
    different_non_trivial_nums ~= -1));
37 different_non_trivial_nums = different_non_trivial_nums(find(
    different_non_trivial_nums ~= 0.5));
different_non_trivial_nums = different_non_trivial_nums(find(
    different_non_trivial_nums ~= -0.5));
39 different_non_trivial_nums = different_non_trivial_nums(find(
    different_non_trivial_nums ~= 0));

41 different_non_trivial_nums = unique(abs(different_non_trivial_nums));
non_trivial = length(abs(different_non_trivial_nums))

43

45 % Jeweils die Menge der verschiedenen Werte ermitteln (hier Re)
num_count_real = zeros(1, length(different_nums_real));
for k = 1:length(different_nums_real)
47     for n = 1:N
49         for m = 1:N
            if different_nums_real(k) == W_r(m,n)
                num_count_real(k) = num_count_real(k) + 1;
51         end
53     end
end

55

57 % Jeweils die Anzahl der verschiedenen Werte ermitteln (hier Im)
num_count_imag = zeros(1, length(different_nums_imag));
59 for k = 1:length(different_nums_imag)
    for n = 1:N

```

```
61     for m = 1:N
62         if different_nums_imag(k) == W_i(m,n)
63             num_count_imag(k) = num_count_imag(k) + 1;
64         end
65     end
66 end
67
68
69 % nicht triviale Werte der reellen Matrix zaehlen
70 nontrivial_nums_real = 0;
71 for k = 1:length(different_nums_real)
72     if abs(different_nums_real(k)) != 1
73         if abs(different_nums_real(k)) != 0.5
74             if different_nums_real(k) != 0
75                 nontrivial_nums_real = nontrivial_nums_real + num_count_real(k);
76             end
77         end
78     end
79 end
80
81 % nicht triviale Werte der imaginaeren Matrix zaehlen
82 nontrivial_nums_imag = 0;
83 for k = 1:length(different_nums_imag)
84     if abs(different_nums_imag(k)) != 1
85         if abs(different_nums_imag(k)) != 0.5
86             if different_nums_imag(k) != 0
87                 nontrivial_nums_imag = nontrivial_nums_imag + num_count_imag(k);
88             end
89         end
90     end
91 end
92
93 nums_of_each_matrix = N*N;
94
95 trivial_nums_real = N*N - nontrivial_nums_real
96 trivial_nums_imag = N*N - nontrivial_nums_imag
97
98 nontrivial_nums_real
99 nontrivial_nums_imag
100
101 trivial_nums_total = trivial_nums_real + trivial_nums_imag
102 nontrivial_nums_total = nontrivial_nums_real + nontrivial_nums_imag
103
104 v = trivial_nums_total/nontrivial_nums_total
105
106 end
```

Listing A.2: Octave-Skript zur Bewertung unterschiedlicher DFT-Twiddlefaktormatrizen

A.2. Twiddlefaktormatrix im S1Q10-Format

```

1 %% Dateiname:      twiddle2file.m
2 %% Funktion:      Erzeugt eine Datei mit den binären komplexen
3 %%                Twiddlefaktoren
4 %% Argumente:      N (Größe der NxN DFT-Matrix)
5 %% Aufbau der Datei: Wie die Matrix, enthält Realteil und Imaginärteil.
6 %%                Alle Werte sind wie im Beispiel durch Leerzeichen
7 %%                getrennt:
8 %%                Re{W(1,1)} Im{W(1,1)} Re{W(1,2)} Im{W(1,2)}
9 %%                Re{W(2,1)} Im{W(2,1)} Re{W(2,2)} Im{W(2,2)}
10 %% Abhängigkeiten: (1) twiddle_coefficients.m
11 %%                (2) dec_to_s1q10.m
12 %%                (3) bit_vector2integer.m
13 %%                (4) zweier_komplement.m
14 %% Author:         Thomas Lattmann
15 %% Datum:          02.11.17
16 %% Version:        1.0

17 function twiddle2file(N)

18
19 % Dezimale Twiddlefaktormatrix erstellen
20 W_dec = twiddle_coefficients(N);
21 W_dec_real = real(W_dec);
22 W_dec_imag = imag(W_dec);
23
24 W_bin_int_real = zeros(size(W_dec_real));
25 W_bin_int_imag = zeros(size(W_dec_imag));
26
27 for m = 1:N
28     for n = 1:N
29         bit_vector = dec_to_s1q10(W_dec_real(m,n));
30         W_bin_int_real(m,n) = bit_vector2integer(bit_vector);
31
32         bit_vector = dec_to_s1q10(W_dec_imag(m,n));
33         W_bin_int_imag(m,n) = bit_vector2integer(bit_vector);
34     end
35 end
36
37 fid=fopen('Twiddle_s1q10_komplex.txt', 'w+');
38
39 for m=1:N
40     for n=1:N
41         fprintf(fid, '%012d ', W_bin_int_real(m,n));
42         fprintf(fid, '%012d', W_bin_int_imag(m,n));
43         if n < N
44             fprintf(fid, ' ');
45         end
46     end
47     if m < N
48         fprintf(fid, '\n');
49     end
50 end

```

```

49     end
    end
51     fclose(fid);
53 end

```

Listing A.3: Erstellen der Twiddlefaktormatrix-Datei

```

%% Dateiname: twiddle_coefficients.m
2 %% Funktion:  Erstellt eine Matrix (W) mit den Twiddlefaktoren fuer die DFT
    der
%%           Groesse, die mit N an das Skript uebergeben wurde.
4 %% Argumente: N (Groesse der NxN DFT-Matrix)
%% Author:    Thomas Lattmann
6 %% Datum:    02.11.17
%% Version:   1.0
8
function W = twiddle_coefficients(N)
10
    % Twiddlefaktoren fuer die DFT
12    W = exp(-i*2*pi*[0:N-1]'.*[0:N-1]/N)
14
    % auf 6 Nachkommastellen reduzieren
    W = round(W*1000000)/1000000;
16
    % negative Nullen auf 0 setzen
18    W_real = real(W);
    W_imag = imag(W);
20    W_real(abs(W_real)<00000.1) = 0;
    W_imag(abs(W_imag)<00000.1) = 0;
22    W = W_real + i*W_imag;
24 end

```

Listing A.4: Erzeugen der Twiddlefaktormatrix

```

%% Dateiname:    dec_to_s1q10.m
2 %% Funktion:    Konvertiert eine Dezimalzahl in das binaere S1Q10-Format
%% Argumente:    Dezimalzahl im Bereich von -2...+2-1/2^10
4 %% Abhaengigkeiten: (1) zweier_komplement.m
%% Author:       Thomas Lattmann
6 %% Datum:       02.11.17
%% Version:      1.0
8
function bit_vector = dec_to_s1q10(val)
10
    bit_width=12;
12    bit_vector=zeros(1,bit_width);
    dec_temp=0;
14    val_abs=abs(val);
    val_int=floor(val_abs);

```

```

16 val_frac=val_abs-val_int;
18 if val > 2-1/2^(bit_width-2) % 1.99902... bei 12 Bit und somit 10 Bit
    fuer Nachkomma
    disp('Diese Zahl kann nicht im s1q11-Format dargestellt werden.')
20 elseif val < -2
    disp('Diese Zahl kann nicht im s1q11-Format dargestellt werden.')
22 else

24     % Vorkommastellen
    if abs(val) >= 1
26         bit_vector(2) = 1;
        if val == -2
28             bit_vector(1) = 1;
        end
30    end

32    % Nachkommastellen
    for k = 1:bit_width-2
34        % berechnen der Differenz des Twiddlefaktors und des derzeitigen
        Wertes der Binaerzahl
        d = val_frac - dec_temp;
36        if d >= 1/2^k
            bit_vector(k+2) = 1;
38            dec_temp = dec_temp+1/2^k;
        end
40    end

42    % 2er-Komplement bilden , falls val negativ
    if val < 0
44        bit_vector=zweier_komplement(bit_vector);
    end
46 end
end

```

Listing A.5: Dezimalzahl nach S1Q10 konvertieren

```

1 %% Dateiname: zweier_komplement.m
  %% Funktion:  Bilden des 2er-Komplements eines "Bit"-Vektors
3 %% Argumente: Vektor aus Nullen und Einsen
  %% Author:    Thomas Lattmann
5 %% Datum:     02.11.17
  %% Version:   1.0
7
  function bit_vector = zweier_komplement(bit_vector)
9     bit_width=length(bit_vector);

11     for j = 1:bit_width
        bit_vector(j) = not(bit_vector(j));
13     end
    bit_vector(bit_width) = bit_vector(bit_width) + 1;
15     for j = 1:bit_width-1

```



```

17     if bit_vector(bit_width -j +1) == 2
        bit_vector(bit_width -j +1) = 0;
        bit_vector(bit_width -j) = bit_vector(bit_width -j) + 1;
19     end
    end
21 end

```

Listing A.6: Bildung des 2er-Komplements

```

1 %% Dateiname: bit_vector2integer.m
  %% Funktion: Wandelt einen Vektor von Zahlen in eine einzelne Zahl (
    Integer)
3 %%           Beispiel: [0 1 1 0 0 1] => 11001
  %%           Um fuehrende Nullen zu erhalten muss z.B. printf('%06d',
    Integer)
5 %%           genutzt werden. Hierbei wird vorne mit Nullen aufgefuellt,
    wenn
  %%           'Integer' weniger als 6 stellen hat.
7 %% Argumente: Vektor (aus Nullen und Einsen)
  %% Author:    Thomas Lattmann
9 %% Datum:     02.11.17
  %% Version:   1.0
11
  function bin_int = bit_vector2integer(bit_vector)
13
    bin_int=0;
15    bit_width=length(bit_vector);
17
    % Konvertierung von Vektor nach Integer
    for l = 1:bit_width
19        bin_int = bin_int + bit_vector(bit_width - l + 1)*10^(l-1);
    end
21
  end

```

Listing A.7: Binär-Vektor in Binär-Integer umwandeln

```

  %% Dateiname: s1q10_to_dec.m
2 %% Funktion: Konvertiert eine binaere Zahl im S1Q10-Format als Dezimalzahl
  %% Argumente: Vektor aus Nullen und Einsen
4 %% Author:    Thomas Lattmann
  %% Datum:     02.11.17
6 %% Version:   1.0
8
  function dec = s1q10_to_dec(bit_vector)
10
    % Dezimalzahl aus s1q10 Binaerzahl berechnen
12
    bit_width=length(bit_vector);
    dec = 0;
14
    if bit_vector(1) == 1

```

```
16     dec = -2;
17     if bit_vector(2) == 1
18         dec = -1;
19     end
20 elseif bit_vector(2) == 1
21     dec = 1;
22 end
23
24 for n = 3:bit_width
25     if bit_vector(n) == 1
26         dec = dec + 1/2^(n-2);
27     end
28 end
end
```

Listing A.8: Kontroll-Skript für S1Q10 nach Dezimal

B. Gate-Reports der Syntheseergebnisse

B.1. Gate-Report des 13 Bit Konstantenmultiplizierers

1

Generated by: Encounter(R) RTL Compiler RC14.25 – v14.20–s046_1

3

Generated on: Apr 12 2018 05:17:30 pm

Module: konstantenmultiplizierer

5

Technology library: c35_CORELIB_TYP 3.02

Operating conditions: _nominal_ (balanced_tree)

7

Wireload mode: enclosed

Area mode: timing library

9

11

Gate Instances Area Library

13

ADD21 6 873.600 c35_CORELIB_TYP

15

ADD31 34 9282.000 c35_CORELIB_TYP

AOI210 2 145.600 c35_CORELIB_TYP

17

CLKIN3 2 72.800 c35_CORELIB_TYP

IMUX20 4 364.000 c35_CORELIB_TYP

19

INV2 10 364.000 c35_CORELIB_TYP

MAJ31 4 436.800 c35_CORELIB_TYP

21

NAND20 9 491.400 c35_CORELIB_TYP

NOR20 1 54.600 c35_CORELIB_TYP

23

OAI210 6 436.800 c35_CORELIB_TYP

XNR20 3 327.600 c35_CORELIB_TYP

25

XOR21 1 127.400 c35_CORELIB_TYP

27

total 82 12976.600

29

31

Type Instances Area Area %

33

inverter 12 436.800 3.4

logic 70 12539.800 96.6

35

total 82 12976.600 100.0

Listing B.1: RC Gate-Report des Konstanten multiplizierers

B.2. Gate-Report des 13 Bit Multiplizierers

Generated by:	Encounter(R) RTL Compiler RC14.25 – v14.20–s046_1		
Generated on:	Apr 17 2018 04:51:50 pm		
Module:	multiplizierer		
Technology library:	c35_CORELIB_TYP 3.02		
Operating conditions:	_nominal_ (balanced_tree)		
Wireload mode:	enclosed		
Area mode:	timing library		
Gate	Instances	Area	Library
ADD21	6	873.600	c35_CORELIB_TYP
ADD31	27	7371.000	c35_CORELIB_TYP
AOI220	10	910.000	c35_CORELIB_TYP
CLKIN3	4	145.600	c35_CORELIB_TYP
IMUX20	58	5278.000	c35_CORELIB_TYP
IMUX21	1	91.000	c35_CORELIB_TYP
INV2	11	400.400	c35_CORELIB_TYP
INV3	9	327.600	c35_CORELIB_TYP
MAJ31	8	873.600	c35_CORELIB_TYP
NAND20	6	327.600	c35_CORELIB_TYP
NOR20	13	709.800	c35_CORELIB_TYP
OAI220	32	2912.000	c35_CORELIB_TYP
XNR20	2	218.400	c35_CORELIB_TYP
XNR30	2	400.400	c35_CORELIB_TYP
XNR31	3	600.600	c35_CORELIB_TYP
XNR41	2	546.000	c35_CORELIB_TYP
total	194	21985.600	
Type	Instances	Area	Area %
inverter	24	873.600	4.0
logic	170	21112.000	96.0
total	194	21985.600	100.0

Listing B.2: RC Gate-Report des Multiplizierers

B.3. Gate-Report des 12 Bit Addierers

Generated by:	Encounter(R) RTL Compiler RC14.25 – v14.20–s046_1
---------------	---

```

3  Generated on:      Apr 12 2018  04:14:25 pm
   Module:          Addierer_TOP
5  Technology library: c35_CORELIB_TYP 3.02
   Operating conditions: _nominal_ (balanced_tree)
7  Wireload mode:   enclosed
   Area mode:       timing library

```

Gate	Instances	Area	Library
ADD21	1	145.600	c35_CORELIB_TYP
ADD31	11	3003.000	c35_CORELIB_TYP
CLKIN3	2	72.800	c35_CORELIB_TYP
INV2	1	36.400	c35_CORELIB_TYP
total	15	3257.800	

Type	Instances	Area	Area %
inverter	3	109.200	3.4
logic	12	3148.600	96.6
total	15	3257.800	100.0

Listing B.3: RC Gate-Report des 12 Bit Addierers

B.4. Gate-Report des 13 Bit Negierers

```

1  Generated by:      Encounter(R) RTL Compiler RC14.25 - v14.20-s046_1
3  Generated on:      Apr 12 2018  04:48:10 pm
   Module:          negierer_top
5  Technology library: c35_CORELIB_TYP 3.02
   Operating conditions: _nominal_ (balanced_tree)
7  Wireload mode:   enclosed
   Area mode:       timing library

```

Gate	Instances	Area	Library
ADD21	11	1601.600	c35_CORELIB_TYP
CLKIN2	11	400.400	c35_CORELIB_TYP
INV2	1	36.400	c35_CORELIB_TYP
XNR20	1	109.200	c35_CORELIB_TYP

19	total	24	2147.600	
21				
23	Type	Instances	Area	Area %
25	inverter	12	436.800	20.3
26	logic	12	1710.800	79.7
27	total	24	2147.600	100.0

Listing B.4: RC Gate-Report des 13 Bit Negierers

B.5. Gate-Report der 2D-DFT

1	Generated by: Encounter(R) RTL Compiler RC14.25 – v14.20–s046_1			
3	Generated on: Apr 10 2018 12:04:20 pm			
	Module: dft8optimiert			
5	Technology library: c35_CORELIB_TYP 3.02			
	Operating conditions: _nominal_ (balanced_tree)			
7	Wireload mode: enclosed			
	Area mode: timing library			
9				
11	Gate	Instances	Area	Library
13	ADD21	28	4076.800	c35_CORELIB_TYP
15	ADD31	413	112749.000	c35_CORELIB_TYP
	AOI210	333	24242.401	c35_CORELIB_TYP
17	AOI211	3	218.400	c35_CORELIB_TYP
	AOI2110	70	6370.000	c35_CORELIB_TYP
19	AOI220	2472	224952.000	c35_CORELIB_TYP
	AOI310	17	1547.000	c35_CORELIB_TYP
21	BUF2	1536	83865.597	c35_CORELIB_TYP
	CLKIN2	1	36.400	c35_CORELIB_TYP
23	CLKIN3	124	4513.600	c35_CORELIB_TYP
	DF3	3	819.000	c35_CORELIB_TYP
25	DL3	19	3803.800	c35_CORELIB_TYP
	DLQ1	1	182.000	c35_CORELIB_TYP
27	DLQ3	3231	588042.000	c35_CORELIB_TYP
	IMAJ30	81	8845.200	c35_CORELIB_TYP
29	IMUX20	66	6006.000	c35_CORELIB_TYP
	INV0	478	17399.201	c35_CORELIB_TYP
31	INV12	12	1092.000	c35_CORELIB_TYP
	INV15	14	1528.800	c35_CORELIB_TYP
33	INV2	806	29338.402	c35_CORELIB_TYP
	INV3	175	6370.000	c35_CORELIB_TYP

35	INV6	6	327.600	c35_CORELIB_TYP
	MAJ31	71	7753.200	c35_CORELIB_TYP
37	MUX21	384	41932.799	c35_CORELIB_TYP
	MUX22	5	546.000	c35_CORELIB_TYP
39	NAND20	2222	121321.196	c35_CORELIB_TYP
	NAND21	143	7807.800	c35_CORELIB_TYP
41	NAND22	13	709.800	c35_CORELIB_TYP
	NAND24	4	436.800	c35_CORELIB_TYP
43	NAND28	24	4368.000	c35_CORELIB_TYP
	NAND30	68	4950.400	c35_CORELIB_TYP
45	NAND31	3	218.400	c35_CORELIB_TYP
	NAND40	57	5187.000	c35_CORELIB_TYP
47	NAND41	143	13013.000	c35_CORELIB_TYP
	NAND42	3	436.800	c35_CORELIB_TYP
49	NOR20	460	25115.999	c35_CORELIB_TYP
	NOR21	29	1583.400	c35_CORELIB_TYP
51	NOR22	5	364.000	c35_CORELIB_TYP
	NOR23	1	91.000	c35_CORELIB_TYP
53	NOR24	1	109.200	c35_CORELIB_TYP
	NOR30	36	2620.800	c35_CORELIB_TYP
55	NOR40	18	1310.400	c35_CORELIB_TYP
	OAI210	686	49940.802	c35_CORELIB_TYP
57	OAI2110	199	18109.000	c35_CORELIB_TYP
	OAI212	1	72.800	c35_CORELIB_TYP
59	OAI220	383	34853.000	c35_CORELIB_TYP
	OAI310	28	2548.000	c35_CORELIB_TYP
61	XNR20	309	33742.799	c35_CORELIB_TYP
	XNR30	27	5405.400	c35_CORELIB_TYP
63	XNR31	11	2202.200	c35_CORELIB_TYP
	XOR20	13	1656.200	c35_CORELIB_TYP
65	XOR21	63	8026.200	c35_CORELIB_TYP
	XOR30	10	2002.000	c35_CORELIB_TYP
67	XOR31	1	200.200	c35_CORELIB_TYP
<hr/>				
69	total	15310	1524959.795	
<hr/>				
71				
73	Type	Instances	Area	Area %
<hr/>				
75	sequential	3254	592846.800	38.9
	inverter	1616	60606.003	4.0
77	buffer	1536	83865.597	5.5
	logic	8904	787641.395	51.6
<hr/>				
79	total	15310	1524959.795	100.0

Listing B.5: RC Gate-Report der 2D-DFT

C. VHDL-Code

C.1. Konstantendeklarationen

```
1 library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;

3

5
  package constants is
7     constant mat_size : integer;
     constant bit_width_extern : integer;
9     constant bit_width_adder : integer;
     constant bit_width_multiplier : integer;
11 end constants;

13 package body constants is
     constant mat_size : integer := 8;
15     constant bit_width_extern : integer := 13;
     constant bit_width_adder : integer := bit_width_extern+1;
17     constant bit_width_multiplier : integer := bit_width_adder*2;

19 end constants;
```

Listing C.1: Deklaration der Konstanten

C.2. Datentypendeklarationen

```
— Package, welches ein 2D-Array bereitstellt.
2 — Das 2D-Array besteht aus 1D-Arrays, dies bringr gegenueber der direkten
   Erzeugung (m,n) statt (m)(n) den Vorteil, dass
— dass zeilen- sowie spaltenweise zugewiesen werden kann. Sonst waere nur
   die komplette Matrix oder einzelne Elemente moeglich.

4
  library IEEE;
6  use IEEE.STD_LOGIC_1164.ALL;
  use ieee.numeric_std.all;
8  library work;
  use work.all;
10 use constants.all;

12
  package datatypes is
```



```

14  type t_1d_array is array(integer range 0 to mat_size-1) of signed(
    bit_width_extern-1 downto 0);
    type t_2d_array is array(integer range 0 to mat_size-1) of t_1d_array;

16

    type t_1d_array6_13bit is array(integer range 0 to 5) of signed(
    bit_width_adder-1 downto 0);

18

20  subtype t_twiddle_coeff_long is signed(16 downto 0);
    constant twiddle_coeff_long : t_twiddle_coeff_long := "
00101101010000010";
22  subtype t_twiddle_coeff is signed(bit_width_adder-1 downto 0);
    --constant twiddle_coeff : t_twiddle_coeff := twiddle_coeff_long(16
    downto 16-(bit_width_adder-1));

24

26

28  -- Zustandsautomat 1D-DFT
    subtype t_dft8_states is std_logic_vector(2 downto 0);
30  constant idle          : t_dft8_states := "000";
    constant twiddle_calc   : t_dft8_states := "001";
32  constant additions_stage1 : t_dft8_states := "010";
    constant additions_stage2 : t_dft8_states := "011";
34  constant const_mult      : t_dft8_states := "100";
    constant additions_stage3 : t_dft8_states := "101";
36  constant set_ready_bit    : t_dft8_states := "110";

38 end datatypes;

```

Listing C.2: Deklaration eigener Datentypen

C.3. Testwerte aus Datei einlesen

```

library IEEE;
2 use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

4

  library STD; -- for reading text file
6 use STD.TEXTIO.ALL;
  use ieee.std_logic_textio.all;

8

  library work;
10 use work.all;
  use datatypes.all;
12 use constants.all;

14

entity read_input_matrix is

```

```

16  port(
17      clk          : in  bit;
18      loaded       : out bit;
19      input_real   : out t_2d_array;
20      input_imag   : out t_2d_array
21  );
22  end entity read_input_matrix;

24
25  architecture bhv of read_input_matrix is
26  begin
27      reading : process
28
29          variable element_1_real : std_logic_vector(bit_width_extern-1
30      downto 0) := (others => '0');
31          variable element_1_imag : std_logic_vector(bit_width_extern-1
32      downto 0) := (others => '0');
33          variable element_2_real : std_logic_vector(bit_width_extern-1
34      downto 0) := (others => '0');
35          variable element_2_imag : std_logic_vector(bit_width_extern-1
36      downto 0) := (others => '0');
37          variable element_3_real : std_logic_vector(bit_width_extern-1
38      downto 0) := (others => '0');
39          variable element_3_imag : std_logic_vector(bit_width_extern-1
40      downto 0) := (others => '0');
41          variable element_4_real : std_logic_vector(bit_width_extern-1
42      downto 0) := (others => '0');
43          variable element_4_imag : std_logic_vector(bit_width_extern-1
44      downto 0) := (others => '0');
45          variable element_5_real : std_logic_vector(bit_width_extern-1
46      downto 0) := (others => '0');
47          variable element_5_imag : std_logic_vector(bit_width_extern-1
48      downto 0) := (others => '0');
49          variable element_6_real : std_logic_vector(bit_width_extern-1
50      downto 0) := (others => '0');
51          variable element_6_imag : std_logic_vector(bit_width_extern-1
52      downto 0) := (others => '0');
53          variable element_7_real : std_logic_vector(bit_width_extern-1
54      downto 0) := (others => '0');
55          variable element_7_imag : std_logic_vector(bit_width_extern-1
56      downto 0) := (others => '0');
57          variable element_8_real : std_logic_vector(bit_width_extern-1
58      downto 0) := (others => '0');
59          variable element_8_imag : std_logic_vector(bit_width_extern-1
60      downto 0) := (others => '0');
61
62          variable r_space : character;
63
64          variable fstatus : file_open_status; -- status r,w
65          variable inline : line; -- readout line
66          file infile : text; -- filehandle for reading ascii text

```

```
52     variable textfilename : string(1 to 29);
54
56
58     if bit_width_extern = 12 then
59         textfilename := "InputMatrix_komplex_12Bit.txt";
60     else
61         textfilename := "InputMatrix_komplex_16Bit.txt";
62     end if;
64
65     file_open(fstatus, infile, textfilename, read_mode);
66
67     if fstatus = NAME_ERROR then
68         file_open(fstatus, infile, "HDL/InputMatrix_komplex.txt",
69 read_mode);
70         --report "Ausgabe-Datei befindet sich im Unterverzeichnis 'HDL
71         '.";
72     end if;
73
74     for i in 0 to mat_size-1 loop
75
76         wait until clk = '1' and clk'event;
77         readline(infile, inline);
78         read(inline, element_1_real);
79         read(inline, r_space);
80         read(inline, element_1_imag);
81         read(inline, r_space);
82         read(inline, element_2_real);
83         read(inline, r_space);
84         read(inline, element_2_imag);
85         read(inline, r_space);
86         read(inline, element_3_real);
87         read(inline, r_space);
88         read(inline, element_3_imag);
89         read(inline, r_space);
90         read(inline, element_4_real);
91         read(inline, r_space);
92         read(inline, element_4_imag);
93         read(inline, r_space);
94         read(inline, element_5_real);
95         read(inline, r_space);
96         read(inline, element_5_imag);
97         read(inline, r_space);
98         read(inline, element_6_real);
99         read(inline, r_space);
100        read(inline, element_6_imag);
101        read(inline, r_space);
102        read(inline, element_7_real);
```

```

100     read(inline , r_space);
101     read(inline , element_7_imag);
102     read(inline , r_space);
103     read(inline , element_8_real);
104     read(inline , r_space);
105     read(inline , element_8_imag);
106
107     input_real(i)(0) <= signed(element_1_real);
108     input_imag(i)(0) <= signed(element_1_imag);
109     input_real(i)(1) <= signed(element_2_real);
110     input_imag(i)(1) <= signed(element_2_imag);
111     input_real(i)(2) <= signed(element_3_real);
112     input_imag(i)(2) <= signed(element_3_imag);
113     input_real(i)(3) <= signed(element_4_real);
114     input_imag(i)(3) <= signed(element_4_imag);
115     input_real(i)(4) <= signed(element_5_real);
116     input_imag(i)(4) <= signed(element_5_imag);
117     input_real(i)(5) <= signed(element_6_real);
118     input_imag(i)(5) <= signed(element_6_imag);
119     input_real(i)(6) <= signed(element_7_real);
120     input_imag(i)(6) <= signed(element_7_imag);
121     input_real(i)(7) <= signed(element_8_real);
122     input_imag(i)(7) <= signed(element_8_imag);
123
124     if i = mat_size-1 then
125         loaded <= '1' after 10 ns;
126     end if;
127 end loop;
128 file_close(infile);
129 wait;
130
131 end process;
132 end bhv;

```

Listing C.3: Eingangs-Matrix aus Textdatei einlesen

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  library work;
6  use work.all;
7  use datatypes.all;
8
9  entity read_input_matrix_tb is
10 end entity read_input_matrix_tb;
11
12 architecture arch of read_input_matrix_tb is
13
14     signal clk          : bit := '0';
15     signal loaded       : bit := '0';

```

```
17  signal input_real   : t_2d_array;
    signal input_imag  : t_2d_array;

19  component read_input_matrix is
    port(
21      clk           : in  bit;
        loaded        : out bit;
23      input_real    : out t_2d_array;
        input_imag     : out t_2d_array
25    );
    end component;

27  begin
29      dut : read_input_matrix
        port map(
31          clk           => clk ,
            loaded        => loaded ,
33          input_real    => input_real ,
            input_imag     => input_imag
35        );

37      clk <= not clk after 20 ns;
    end arch;
```

Listing C.4: Testbench für das Einlesen aus einer Textdatei

C.4. Ergebnisse in Datei schreiben

```
1  library IEEE;
    use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;

5  library STD; — for writing text file
    use STD.TEXTIO.ALL;
7  use ieee.std_logic_textio.all;

9  library work;
    use work.all;
11 use datatypes.all;
    use constants.all;

13

15
17 entity write_results is
    port(
19     result_ready : in  bit;
        result_real  : in  t_2d_array;
        result_imag  : in  t_2d_array;
21     write_done   : out bit
```

```

    );
23 end entity write_results;

25
architecture bhv of write_results is
27 begin
    writing_to_file : process(result_ready)
29
        variable fstatus : file_open_status; -- status r,w
31        variable outline : line; -- writeout line
        file        outfile : text; -- filehandle
33
        --variable output1 : bit_vector(3 downto 0) := "0101";
35        --variable output2 : bit_vector(3 downto 0) := "0110";

        variable element_1_real : std_logic_vector(bit_width_extern-1 downto 0)
        ;
        variable element_1_imag : std_logic_vector(bit_width_extern-1 downto 0)
        ;
39        variable element_2_real : std_logic_vector(bit_width_extern-1 downto 0)
        ;
        variable element_2_imag : std_logic_vector(bit_width_extern-1 downto 0)
        ;
41        variable element_3_real : std_logic_vector(bit_width_extern-1 downto 0)
        ;
        variable element_3_imag : std_logic_vector(bit_width_extern-1 downto 0)
        ;
43        variable element_4_real : std_logic_vector(bit_width_extern-1 downto 0)
        ;
        variable element_4_imag : std_logic_vector(bit_width_extern-1 downto 0)
        ;
45        variable element_5_real : std_logic_vector(bit_width_extern-1 downto 0)
        ;
        variable element_5_imag : std_logic_vector(bit_width_extern-1 downto 0)
        ;
47        variable element_6_real : std_logic_vector(bit_width_extern-1 downto 0)
        ;
        variable element_6_imag : std_logic_vector(bit_width_extern-1 downto 0)
        ;
49        variable element_7_real : std_logic_vector(bit_width_extern-1 downto 0)
        ;
        variable element_7_imag : std_logic_vector(bit_width_extern-1 downto 0)
        ;
51        variable element_8_real : std_logic_vector(bit_width_extern-1 downto 0)
        ;
        variable element_8_imag : std_logic_vector(bit_width_extern-1 downto 0)
        ;
53        variable space : character := ' ';

55        begin

```

```

57     file_open(fstatus , outfile , "/home/tlattmann/cadence/mat_mult/HDL/
Results.txt" , write_mode);

59     --if result_ready = '1' then

61     for i in 0 to mat_size-1 loop
        element_1_real := std_logic_vector(result_real(i)(0));
63         element_1_imag := std_logic_vector(result_imag(i)(0));
        element_2_real := std_logic_vector(result_real(i)(1));
65         element_2_imag := std_logic_vector(result_imag(i)(1));
        element_3_real := std_logic_vector(result_real(i)(2));
67         element_3_imag := std_logic_vector(result_imag(i)(2));
        element_4_real := std_logic_vector(result_real(i)(3));
69         element_4_imag := std_logic_vector(result_imag(i)(3));
        element_5_real := std_logic_vector(result_real(i)(4));
71         element_5_imag := std_logic_vector(result_imag(i)(4));
        element_6_real := std_logic_vector(result_real(i)(5));
73         element_6_imag := std_logic_vector(result_imag(i)(5));
        element_7_real := std_logic_vector(result_real(i)(6));
75         element_7_imag := std_logic_vector(result_imag(i)(6));
        element_8_real := std_logic_vector(result_real(i)(7));
77         element_8_imag := std_logic_vector(result_imag(i)(7));

79         write(outline , element_1_real);
        write(outline , space);
81         write(outline , element_1_imag);
        write(outline , space);
83         write(outline , element_2_real);
        write(outline , space);
85         write(outline , element_2_imag);
        write(outline , space);
87         write(outline , element_3_real);
        write(outline , space);
89         write(outline , element_3_imag);
        write(outline , space);
91         write(outline , element_4_real);
        write(outline , space);
93         write(outline , element_4_imag);
        write(outline , space);
95         write(outline , element_5_real);
        write(outline , space);
97         write(outline , element_5_imag);
        write(outline , space);
99         write(outline , element_6_real);
        write(outline , space);
101        write(outline , element_6_imag);
        write(outline , space);
103        write(outline , element_7_real);
        write(outline , space);
105        write(outline , element_7_imag);
        write(outline , space);

```

```

107     write(outline , element_8_real);
        write(outline , space);
109     write(outline , element_8_imag);

111     writeline(outfile , outline);
end loop;
113
        write_done <= '1';
115     file_close(outfile);
    —end if;
117
end process;
119 end bhv;

```

Listing C.5: Ergebnis-Matrix in Textdatei schreiben

```

1  library IEEE;
   use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;

5  library STD; — for writing text file
   use STD.TEXTIO.ALL;
7  use ieee.std_logic_textio.all;

9  library work;
   use work.all;
11 use datatypes.all;
   use constants.all;
13

15 entity write_test_tb is
end entity write_test_tb;
17

19 architecture bhv of write_test_tb is

21     signal clk          : bit;
   signal loaded          : bit;
23     signal result_ready : bit;
   signal write_done      : bit;
25     signal loop_running : bit;
   signal loop_number     : signed(2 downto 0);
27     signal input_real   : t_2d_array;
   signal input_imag     : t_2d_array;
29     signal output       : std_logic_vector(bit_width_extern-1 downto 0);

31     component read_input_matrix
       port(
33         clk          : in  bit;
         loaded        : out bit;
35         input_real   : out t_2d_array;
         input_imag    : out t_2d_array

```



```

37     );
38 end component;
39
40 component write_results
41     port(
42         result_ready : in bit;
43         result_real   : in t_2d_array;
44         result_imag   : in t_2d_array;
45         write_done    : out bit;
46         loop_number   : out signed(2 downto 0);
47         loop_running  : out bit;
48         output        : out std_logic_vector(bit_width_extern-1 downto 0)
49     );
50 end component;
51
52 begin
53
54     mat : read_input_matrix
55         port map(
56             clk          => clk ,
57             loaded       => loaded ,
58             input_real   => input_real ,
59             input_imag   => input_imag
60         );
61
62     write : write_results
63         port map(
64             result_ready => result_ready ,
65             result_real  => input_real ,
66             result_imag  => input_imag ,
67             write_done   => write_done ,
68             loop_number  => loop_number ,
69             loop_running => loop_running ,
70             output       => output
71         );
72
73     result_ready <= loaded after 20 ns;
74     clk          <= not clk after 10 ns;
75
76 end bhv;

```

Listing C.6: Testbench für das schreiben in eine Textdatei

C.5. Berechnung der 2D-DFT

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use ieee.numeric_std.all;
4 library work;

```

```

use work.all;
6 use datatypes.all;
use constants.all;
8
10 library STD; -- for reading text file
use STD.TEXTIO.ALL;
12 use ieee.std_logic_textio.all;

14 entity dft8optimiert is
port(
16     clk           : in  bit;
    nReset         : in  bit;
18     loaded        : in  bit;
    input_real      : in  t_2d_array;
20     input_imag     : in  t_2d_array;
    result_real     : out t_2d_array;
22     result_imag    : out t_2d_array;
    result_ready    : out bit;
24     idft          : in  bit;
    state_out       : out t_dft8_states;
26     element_out    : out unsigned(5 downto 0);
    dft_1d_2d_out   : out bit
28 );
end dft8optimiert;
30

32 architecture arch of dft8optimiert is

34     signal dft_state, next_dft_state : t_dft8_states;

36
begin
38     FSM_TAKT: process(clk)
begin
40         if clk='1' and clk'event then
42             dft_state <= dft_state;
            state_out <= dft_state;
44             if nReset='0' then
                dft_state <= idle;
46                 state_out <= idle;
            elsif loaded = '0' then
48                 dft_state <= idle;
                state_out <= idle;
50             elsif loaded='1' and dft_state = idle then
                dft_state <= twiddle_calc;
52                 state_out <= twiddle_calc;
            else
54                 dft_state <= next_dft_state;
                state_out <= next_dft_state;

```

```

56     end if;
57     end if;
58 end process;

60
61 FSM_KOMB: process(dft_state)
62     --constant twiddle_coeff : signed(16 downto 0) := "00010110101000001";
63     variable twiddle_coeff : signed(bit_width_adder-1 downto 0);
64
65     variable mult_re, mult_im : signed(bit_width_multiplier-1 downto 0);
66
67     variable W_row, l_col : integer;
68     variable dft_1d_real, dft_1d_imag : t_2d_array;
69     variable matrix_real, matrix_imag : t_2d_array;
70     variable temp_re, temp_im : t_1d_array6_13bit;
71     variable temp14bit_re, temp14bit_im : signed(bit_width_adder downto 0);
72     variable dft_1d_2d : bit;
73     variable element : unsigned(5 downto 0) := "000000";
74
75
76     variable row_col_idx : integer := 0;
77
78     --variable LineBuffer : LINE;
79
80
81 begin
82     twiddle_coeff := "0001011010100";
83     -- Flip-Flops
84     -- werden das 1. Mal sich selbst zu gewiesen, bevor sie einen Wert
85     haben!
86     result_ready <= '0';
87     element := element;
88     dft_1d_2d := dft_1d_2d;
89     temp_re := temp_re;
90     temp_im := temp_im;
91     mult_re := mult_re;
92     mult_im := mult_im;
93     dft_1d_real := dft_1d_real;
94     dft_1d_imag := dft_1d_imag;
95     matrix_real := matrix_real;
96     matrix_imag := matrix_imag;
97     dft_1d_2d_out <= dft_1d_2d;
98
99     -- Die Matrix hat 64 Elemente -> 2^6=64 -> 6-Bit Vektor passt genau.
100    Ueberlauf = 1. Element vom naechsten Durchlauf.
101    -- Der Elemente-Vektor kann darueber hinaus in vordere Haelfte = Zeile
102    und hintere Haelfte = Spalte aufgeteilt werden.
103    -- So laesst sich auch ein Matrix-Element mit zwei Indizes ansprechen:

```

```

104  — Bei der IDFT sind die Zeilen 1 und 7, 2 und 6, 3 und 5 vertauscht. 1
    und 4 bleiben wie sie sind.

106  row_col_idx := to_integer(element(5 downto 3)); — Wird bei der
    Twiddlefaktor-Matrix als Zeilen-, bei der Zwischen- und
    — Ausgangsmatrix als
    Spaltenindex verwendet.

108  if idft = '1' then
110      if row_col_idx = 0 then
        W_row := 0;
112      else
        W_row := 8-row_col_idx; — Twiddlefaktor-Matrix
        end if;
114  else
        W_row := row_col_idx; — Twiddlefaktor-Matrix
116  end if;

118  l_col := to_integer(element(2 downto 0)); — Input-Matrix

120
122  if element = "000000" then
        if dft_1d_2d = '0' then
124            matrix_real := input_real;
            matrix_imag := input_imag;
126        else
            matrix_real := dft_1d_real;
            matrix_imag := dft_1d_imag;
128        end if;
    end if;
130
132  case dft_state is
    when idle =>
134      next_dft_state <= twiddle_calc;

136      when twiddle_calc => — dft_state_out = 1
        — Mit resize werden die 12 Bit Eingangswerte vorzeichengerecht auf
        13 Bit erweitert, um um die richtige Groesse zu haben.
138      — Bei der Addition muessen die Summanden die gleiche Bit-Breite
        wie der Ergebnis-Vektor haben.
        case W_row is
140            — Die Faktoren (Koeffizienten) der Twiddlefaktor-Matrix W lassen
            sich ueber  $\exp(-i \cdot 2 \cdot \pi \cdot [0:7]' \cdot [0:7]/8)$  berechnen.
            — 1. Zeile aus W -> nur Additionen
142            when 0 =>
                — Die 1. Zeile aus W besteht nur aus den Faktoren (1+j0).
                Daraus resultiert, dass die reellen
144            — und die imaginaeren Werte der Eingangs-Matrix unabhaengig
                von einander aufsummiert werden.
                — Real

```

```

146      temp_re(0) := resize(matrix_real(0)(l_col), bit_width_adder)
+ resize(matrix_real(1)(l_col), bit_width_adder);
      temp_re(1) := resize(matrix_real(2)(l_col), bit_width_adder)
148 + resize(matrix_real(3)(l_col), bit_width_adder);
      temp_re(2) := resize(matrix_real(4)(l_col), bit_width_adder)
+ resize(matrix_real(5)(l_col), bit_width_adder);
      temp_re(3) := resize(matrix_real(6)(l_col), bit_width_adder)
+ resize(matrix_real(7)(l_col), bit_width_adder);
150      -- Imag
      temp_im(0) := resize(matrix_imag(0)(l_col), bit_width_adder)
+ resize(matrix_imag(1)(l_col), bit_width_adder);
152      temp_im(1) := resize(matrix_imag(2)(l_col), bit_width_adder)
+ resize(matrix_imag(3)(l_col), bit_width_adder);
      temp_im(2) := resize(matrix_imag(4)(l_col), bit_width_adder)
+ resize(matrix_imag(5)(l_col), bit_width_adder);
154      temp_im(3) := resize(matrix_imag(6)(l_col), bit_width_adder)
+ resize(matrix_imag(7)(l_col), bit_width_adder);

156      -- 2. Zeile aus W besteht aus den Faktoren
158      -- 0: ( 1.00000 + 0.00000i), 1: ( 0.70711 + 0.70711i), 2:
(0.00000 + 1.00000i), 3: (-0.70711 + 0.70711i),
      -- 4: (-1.00000 + 0.00000i), 5: (-0.70711 - 0.70711i), 6:
(0.00000 - 1.00000i), 7: ( 0.70711 - 0.70711i)

160      -- Wegen der Faktoren (+/-0.70711 +/-0.70711i) haben die geraden
Zeilen (beginnend bei 1) 12 statt 8 Subtraktionen
162      -- Zunaechst werden die Werte aufsummiert, die mit dem Faktor 1 "
multipliziert" werden muessen.
      -- Dann werden die Werte aufsummiert, die mit 0,70711
multipliziert werden muessen. Um sowohl den Quelltext und
164      -- insbesondere auch den Platzbedarf auf dem Chip klein zuhalten,
wird die Multiplikation auf die Summe aller und
      -- nicht auf die einzelnen Werte angewandt.
166      -- Da immer genau die Haelfte der Faktoren positiv und die andere
negativ ist, werden die Eingangswerte so sortiert,
      -- dass keine Negationen noetig sind.
168      when 1 =>
      -- Real
170      temp_re(0) := resize(matrix_real(0)(l_col), bit_width_adder)
- resize(matrix_real(4)(l_col), bit_width_adder);
      temp_re(1) := resize(matrix_imag(2)(l_col), bit_width_adder)
- resize(matrix_imag(6)(l_col), bit_width_adder);
172      -- MultPart
      temp_re(2) := resize(matrix_real(1)(l_col), bit_width_adder)
- resize(matrix_real(3)(l_col), bit_width_adder);
174      temp_re(3) := resize(matrix_imag(1)(l_col), bit_width_adder)
- resize(matrix_imag(7)(l_col), bit_width_adder);
      temp_re(4) := resize(matrix_imag(3)(l_col), bit_width_adder)
- resize(matrix_real(5)(l_col), bit_width_adder);

```

```

176         temp_re(5) := resize(matrix_real(7)(l_col), bit_width_adder)
    - resize(matrix_imag(5)(l_col), bit_width_adder);
    - -- Imag
178         temp_im(0) := resize(matrix_imag(0)(l_col), bit_width_adder)
    - resize(matrix_real(2)(l_col), bit_width_adder);
        temp_im(1) := resize(matrix_real(6)(l_col), bit_width_adder)
    - resize(matrix_imag(4)(l_col), bit_width_adder);
180         -- MultPart
        temp_im(2) := resize(matrix_imag(1)(l_col), bit_width_adder)
    - resize(matrix_real(1)(l_col), bit_width_adder);
182         temp_im(3) := resize(matrix_real(5)(l_col), bit_width_adder)
    - resize(matrix_real(3)(l_col), bit_width_adder);
        temp_im(4) := resize(matrix_real(7)(l_col), bit_width_adder)
    - resize(matrix_imag(3)(l_col), bit_width_adder);
184         temp_im(5) := resize(matrix_imag(7)(l_col), bit_width_adder)
    - resize(matrix_imag(5)(l_col), bit_width_adder);

186         -- 3. Zeile aus W
        -- 0: (1.00000 + 0.00000i), 1: (0.00000 + 1.00000i), 2: (-1.00000
    + 0.00000i), 3: (-0.00000 - 1.00000i),
188         -- 4: (1.00000 - 0.00000i), 5: (0.00000 + 1.00000i), 6: (-1.00000
    + 0.00000i), 7: (-0.00000 - 1.00000i)
        when 2 =>
190         -- Real
        temp_re(0) := resize(matrix_real(0)(l_col), bit_width_adder)
    - resize(matrix_real(2)(l_col), bit_width_adder);
192         temp_re(1) := resize(matrix_imag(1)(l_col), bit_width_adder)
    - resize(matrix_imag(3)(l_col), bit_width_adder);
        temp_re(2) := resize(matrix_real(4)(l_col), bit_width_adder)
    - resize(matrix_real(6)(l_col), bit_width_adder);
194         temp_re(3) := resize(matrix_imag(5)(l_col), bit_width_adder)
    - resize(matrix_imag(7)(l_col), bit_width_adder);
        -- Imag
196         temp_im(0) := resize(matrix_imag(0)(l_col), bit_width_adder)
    - resize(matrix_real(1)(l_col), bit_width_adder);
        temp_im(1) := resize(matrix_real(3)(l_col), bit_width_adder)
    - resize(matrix_imag(2)(l_col), bit_width_adder);
198         temp_im(2) := resize(matrix_imag(4)(l_col), bit_width_adder)
    - resize(matrix_real(5)(l_col), bit_width_adder);
        temp_im(3) := resize(matrix_real(7)(l_col), bit_width_adder)
    - resize(matrix_imag(6)(l_col), bit_width_adder);

200         -- 4. Zeile aus W
202         -- 0: ( 1.00000 + 0.00000i), 1: (-0.70711 + 0.70711i), 2:
    (-0.00000 - 1.00000i), 3: ( 0.70711 + 0.70711i)
        -- 4: (-1.00000 + 0.00000i), 5: ( 0.70711 - 0.70711i), 6: (
    0.00000 + 1.00000i), 7: (-0.70711 - 0.70711i)
204         when 3 =>
        -- Real
206         temp_re(0) := resize(matrix_real(0)(l_col), bit_width_adder)
    - resize(matrix_imag(2)(l_col), bit_width_adder);

```

```

temp_re(1) := resize(matrix_imag(6)(l_col), bit_width_adder)
- resize(matrix_real(4)(l_col), bit_width_adder);
208      --MultPart
temp_re(2) := resize(matrix_imag(1)(l_col), bit_width_adder)
- resize(matrix_real(1)(l_col), bit_width_adder);
210      temp_re(3) := resize(matrix_real(3)(l_col), bit_width_adder)
- resize(matrix_imag(5)(l_col), bit_width_adder);
temp_re(4) := resize(matrix_imag(3)(l_col), bit_width_adder)
- resize(matrix_imag(7)(l_col), bit_width_adder);
212      temp_re(5) := resize(matrix_real(5)(l_col), bit_width_adder)
- resize(matrix_real(7)(l_col), bit_width_adder);

214      -- Imag
temp_im(0) := resize(matrix_imag(0)(l_col), bit_width_adder)
- resize(matrix_imag(4)(l_col), bit_width_adder);
216      temp_im(1) := resize(matrix_real(2)(l_col), bit_width_adder)
- resize(matrix_real(6)(l_col), bit_width_adder);
      --MultPart
218      temp_im(2) := resize(matrix_imag(3)(l_col), bit_width_adder)
- resize(matrix_real(1)(l_col), bit_width_adder);
temp_im(3) := resize(matrix_real(5)(l_col), bit_width_adder)
- resize(matrix_imag(1)(l_col), bit_width_adder);
220      temp_im(4) := resize(matrix_imag(5)(l_col), bit_width_adder)
- resize(matrix_real(3)(l_col), bit_width_adder);
temp_im(5) := resize(matrix_real(7)(l_col), bit_width_adder)
- resize(matrix_imag(7)(l_col), bit_width_adder);
222

      -- 5. Zeile
224      -- 0: (1.00000 + 0.00000i), 1: (-1.00000 + 0.00000i), 2: (1.00000
- 0.00000i), 3: (-1.00000 + 0.00000i),
      -- 4: (1.00000 - 0.00000i), 5: (-1.00000 + 0.00000i), 6: (1.00000
- 0.00000i), 7: (-1.00000 + 0.00000i)
226      when 4 =>
      -- Real
228      temp_re(0) := resize(matrix_real(0)(l_col), bit_width_adder)
- resize(matrix_real(1)(l_col), bit_width_adder);
temp_re(1) := resize(matrix_real(2)(l_col), bit_width_adder)
- resize(matrix_real(3)(l_col), bit_width_adder);
230      temp_re(2) := resize(matrix_real(4)(l_col), bit_width_adder)
- resize(matrix_real(5)(l_col), bit_width_adder);
temp_re(3) := resize(matrix_real(6)(l_col), bit_width_adder)
- resize(matrix_real(7)(l_col), bit_width_adder);
232      -- Imag
temp_im(0) := resize(matrix_imag(0)(l_col), bit_width_adder)
- resize(matrix_imag(1)(l_col), bit_width_adder);
234      temp_im(1) := resize(matrix_imag(2)(l_col), bit_width_adder)
- resize(matrix_imag(3)(l_col), bit_width_adder);
temp_im(2) := resize(matrix_imag(4)(l_col), bit_width_adder)
- resize(matrix_imag(5)(l_col), bit_width_adder);
236      temp_im(3) := resize(matrix_imag(6)(l_col), bit_width_adder)
- resize(matrix_imag(7)(l_col), bit_width_adder);

```

```

238      -- 6. Zeile
      -- 0: ( 1.00000 + 0.00000i), 1: (-0.70711 - 0.70711i), 2: (
240      0.00000 + 1.00000i), 3: ( 0.70711 - 0.70711i),
      -- 4: (-1.00000 + 0.00000i) 5: ( 0.70711 + 0.70711i), 6:
242      (-0.00000 - 1.00000i), 7: (-0.70711 + 0.70711i)
      when 5 =>
      -- Real
      temp_re(0) := resize(matrix_real(0)(l_col), bit_width_adder)
      -- resize(matrix_real(4)(l_col), bit_width_adder);
244      temp_re(1) := resize(matrix_imag(2)(l_col), bit_width_adder)
      -- resize(matrix_imag(6)(l_col), bit_width_adder);
      --MultPart
246      temp_re(2) := resize(matrix_real(3)(l_col), bit_width_adder)
      -- resize(matrix_real(1)(l_col), bit_width_adder);
      temp_re(3) := resize(matrix_real(5)(l_col), bit_width_adder)
      -- resize(matrix_imag(1)(l_col), bit_width_adder);
248      temp_re(4) := resize(matrix_imag(5)(l_col), bit_width_adder)
      -- resize(matrix_imag(3)(l_col), bit_width_adder);
      temp_re(5) := resize(matrix_imag(7)(l_col), bit_width_adder)
      -- resize(matrix_real(7)(l_col), bit_width_adder);
250      -- Imag
      temp_im(0) := resize(matrix_imag(0)(l_col), bit_width_adder)
      -- resize(matrix_real(2)(l_col), bit_width_adder);
252      temp_im(1) := resize(matrix_real(6)(l_col), bit_width_adder)
      -- resize(matrix_imag(4)(l_col), bit_width_adder);
      --MultPart
254      temp_im(2) := resize(matrix_real(1)(l_col), bit_width_adder)
      -- resize(matrix_imag(1)(l_col), bit_width_adder);
      temp_im(3) := resize(matrix_real(3)(l_col), bit_width_adder)
      -- resize(matrix_real(5)(l_col), bit_width_adder);
256      temp_im(4) := resize(matrix_imag(3)(l_col), bit_width_adder)
      -- resize(matrix_real(7)(l_col), bit_width_adder);
      temp_im(5) := resize(matrix_imag(5)(l_col), bit_width_adder)
      -- resize(matrix_imag(7)(l_col), bit_width_adder);
258
      -- 7. Zeile
      -- 0: (1.00000 + 0.00000i), 1: (-0.00000 - 1.00000i), 2:
260      (-1.00000 + 0.00000i), 3: ( 0.00000 + 1.00000i),
      -- 4: (1.00000 - 0.00000i), 5: (-0.00000 - 1.00000i), 6:
      (-1.00000 + 0.00000i), 7: (-0.00000 + 1.00000i)
262      when 6 =>
      -- Real
264      temp_re(0) := resize(matrix_real(0)(l_col), bit_width_adder)
      -- resize(matrix_imag(1)(l_col), bit_width_adder);
      temp_re(1) := resize(matrix_imag(3)(l_col), bit_width_adder)
      -- resize(matrix_real(2)(l_col), bit_width_adder);
266      temp_re(2) := resize(matrix_real(4)(l_col), bit_width_adder)
      -- resize(matrix_imag(5)(l_col), bit_width_adder);
      temp_re(3) := resize(matrix_imag(7)(l_col), bit_width_adder)
      -- resize(matrix_real(6)(l_col), bit_width_adder);

```



```

268      -- Imag
      temp_im(0) := resize(matrix_imag(0)(l_col), bit_width_adder)
      -- resize(matrix_imag(2)(l_col), bit_width_adder);
270      temp_im(1) := resize(matrix_real(1)(l_col), bit_width_adder)
      -- resize(matrix_real(3)(l_col), bit_width_adder);
      temp_im(2) := resize(matrix_imag(4)(l_col), bit_width_adder)
      -- resize(matrix_imag(6)(l_col), bit_width_adder);
272      temp_im(3) := resize(matrix_real(5)(l_col), bit_width_adder)
      -- resize(matrix_real(7)(l_col), bit_width_adder);

274      -- 8. Zeile
      -- 0: ( 1.00000 + 0.00000i), 1: ( 0.70711 - 0.70711i), 2:
      -- (-0.00000 - 1.00000i), 3: (-0.70711 - 0.70711i),
276      -- 4: (-1.00000 + 0.00000i), 5: (-0.70711 + 0.70711i), 6:
      -- (-0.00000 + 1.00000i), 7: ( 0.70711 + 0.70711i)
      when 7 =>
278          -- Real
          temp_re(0) := resize(matrix_real(0)(l_col), bit_width_adder)
          -- resize(matrix_imag(2)(l_col), bit_width_adder);
280          temp_re(1) := resize(matrix_imag(6)(l_col), bit_width_adder)
          -- resize(matrix_real(4)(l_col), bit_width_adder);
          -- MultPart
282          temp_re(2) := resize(matrix_real(1)(l_col), bit_width_adder)
          -- resize(matrix_imag(1)(l_col), bit_width_adder);
          temp_re(3) := resize(matrix_imag(5)(l_col), bit_width_adder)
          -- resize(matrix_real(3)(l_col), bit_width_adder);
284          temp_re(4) := resize(matrix_real(7)(l_col), bit_width_adder)
          -- resize(matrix_imag(3)(l_col), bit_width_adder);
          temp_re(5) := resize(matrix_imag(7)(l_col), bit_width_adder)
          -- resize(matrix_real(5)(l_col), bit_width_adder);
286          -- Imag
          temp_im(0) := resize(matrix_imag(0)(l_col), bit_width_adder)
          -- resize(matrix_imag(4)(l_col), bit_width_adder);
288          temp_im(1) := resize(matrix_real(2)(l_col), bit_width_adder)
          -- resize(matrix_real(6)(l_col), bit_width_adder);
          -- MultPart
290          temp_im(2) := resize(matrix_real(1)(l_col), bit_width_adder)
          -- resize(matrix_imag(3)(l_col), bit_width_adder);
          temp_im(3) := resize(matrix_imag(1)(l_col), bit_width_adder)
          -- resize(matrix_real(5)(l_col), bit_width_adder);
292          temp_im(4) := resize(matrix_real(3)(l_col), bit_width_adder)
          -- resize(matrix_imag(5)(l_col), bit_width_adder);
          temp_im(5) := resize(matrix_imag(7)(l_col), bit_width_adder)
          -- resize(matrix_real(7)(l_col), bit_width_adder);

294          when others => element := element; -- "dummy arbeit", es sind
          bereits alle Faelle abgedeckt!
296      end case;

298      next_dft_state <= additions_stagel;

```

```

300     when additions_stagel => — dft_state_out = 2
302
303         — Es wird vor jeder Addition ein Bitshift auf die Summanden
        angewandt, um den Wertebereich der Speichervariable beim
        zurueckschreiben nicht zu ueberschreiten (1. Mal)
304
305         — Zeilen 1, 3, 5, 7 (ungerade) aufsummieren (bzw. 0(000XXX), 2(010
        XXX), 4(100XXX), 6(110XXX) beginnend bei 0)
306         if element(3) = '0' then
307
308
309
310             — Real
            temp_re(0) := resize(temp_re(0)(bit_width_adder-1 downto 1),
            bit_width_adder) + resize(temp_re(1)(bit_width_adder-1 downto 1),
            bit_width_adder);
312             temp_re(1) := resize(temp_re(2)(bit_width_adder-1 downto 1),
            bit_width_adder) + resize(temp_re(3)(bit_width_adder-1 downto 1),
            bit_width_adder);
            — Imag
314             temp_im(0) := resize(temp_im(0)(bit_width_adder-1 downto 1),
            bit_width_adder) + resize(temp_im(1)(bit_width_adder-1 downto 1),
            bit_width_adder);
            temp_im(1) := resize(temp_im(2)(bit_width_adder-1 downto 1),
            bit_width_adder) + resize(temp_im(3)(bit_width_adder-1 downto 1),
            bit_width_adder);
316         else
            — gerade Zeilen aus W
318             — Real
            —ConstPart
320             temp_re(0) := resize(temp_re(0)(bit_width_adder-1 downto 1),
            bit_width_adder) + resize(temp_re(1)(bit_width_adder-1 downto 1),
            bit_width_adder);
            —MultPart
322             temp_re(2) := resize(temp_re(2)(bit_width_adder-1 downto 1),
            bit_width_adder) + resize(temp_re(3)(bit_width_adder-1 downto 1),
            bit_width_adder);
            temp_re(4) := resize(temp_re(4)(bit_width_adder-1 downto 1),
            bit_width_adder) + resize(temp_re(5)(bit_width_adder-1 downto 1),
            bit_width_adder);
324             — Imag
            —ConstPart
326             temp_im(0) := resize(temp_im(0)(bit_width_adder-1 downto 1),
            bit_width_adder) + resize(temp_im(1)(bit_width_adder-1 downto 1),
            bit_width_adder);
            —MultPart
328             temp_im(2) := resize(temp_im(2)(bit_width_adder-1 downto 1),
            bit_width_adder) + resize(temp_im(3)(bit_width_adder-1 downto 1),
            bit_width_adder);

```

```

temp_im(4) := resize(temp_im(4)(bit_width_adder-1 downto 1),
bit_width_adder) + resize(temp_im(5)(bit_width_adder-1 downto 1),
bit_width_adder);
330     end if;

332     next_dft_state <= additions_stage2;

334

336     when additions_stage2 => — dft_state_out = 3
        — Es wird vor jeder Addition ein Bitshift auf die Summanden
        angewandt, um den Wertebereich der Speichervariable nicht zu
        ueberschreiten (2. Mal)
        — Zusaetzlich wird beim Zuweisen der ungeraden Zeilen an die
        1D-DFT-Matrix zwei weitere Male geshiftet.
338        — 1 Mal, um den Wertebereich der 1D- bzw. 2D-DFT-Matrix klein
        genug zu halten, ein weiteres Mal, um gleich oft wie bei den geraden
        Zeilen zu shiften

340        — Zeilen 1, 3, 5, 7 (wie oben)
        if element(3) = '0' then
342
344            — Real
            temp_re(0) := resize(temp_re(0)(bit_width_adder-1 downto 1),
            bit_width_adder) + resize(temp_re(1)(bit_width_adder-1 downto 1),
            bit_width_adder);
            — Imag
346            temp_im(0) := resize(temp_im(0)(bit_width_adder-1 downto 1),
            bit_width_adder) + resize(temp_im(1)(bit_width_adder-1 downto 1),
            bit_width_adder);

348            — Hier werden die Bits um 2 Stellen nach rechts geschoben,
            damit die Werte mit den Zeilen 2, 4, 6, 8 vergleichbar sind. Dort wird
            insgesamt gleich
            — oft geshiftet, aber auch 1x mehr aufaddiert.
            — Indizes vertauschen -> Transponiert abspeichern
350            if dft_1d_2d = '0' then
352                dft_1d_real(l_col)(row_col_idx) := resize(temp_re(0)(
            bit_width_adder-1 downto 2), bit_width_extern);
                dft_1d_imag(l_col)(row_col_idx) := resize(temp_im(0)(
            bit_width_adder-1 downto 2), bit_width_extern);
354            else
                result_real(l_col)(row_col_idx) <= resize(temp_re(0)(
            bit_width_adder-1 downto 2), bit_width_extern);
356                result_imag(l_col)(row_col_idx) <= resize(temp_im(0)(
            bit_width_adder-1 downto 2), bit_width_extern);
            end if;

358            element := element+1;
360            element_out <= element;

362            — naechster Zustand

```

```

next_dft_state <= twiddle_calc;

364
else
366     -- Real
    temp_re(2) := resize(temp_re(2)(bit_width_adder-1 downto 1),
bit_width_adder) + resize(temp_re(4)(bit_width_adder-1 downto 1),
bit_width_adder);

368
    -- Imag
370    temp_im(2) := resize(temp_im(2)(bit_width_adder-1 downto 1),
bit_width_adder) + resize(temp_im(4)(bit_width_adder-1 downto 1),
bit_width_adder);

372
    -- naechster Zustand
    next_dft_state <= const_mult;
374
end if;

376

when const_mult => -- dft_state_out = 4
378
    -- Der Zielvektor der Multiplikation ist 26 Bit breit, die beiden
    Multiplikanten sind mit je 13 Bit wie gefordert halb so breit.

380
    -- Zeilen 2, 4, 6, 8 (vergleichbar mit oben)
382    mult_re := temp_re(2) * twiddle_coeff; --(16 downto 16-(
bit_width_adder-1));
    mult_im := temp_im(2) * twiddle_coeff; --(16 downto 16-(
bit_width_adder-1));

384
    next_dft_state <= additions_stage3;

386

when additions_stage3 => -- dft_state_out = 5
388
    -- Die vordersten 12 Bit des Multiplikationsergebnisses werden
    verwendet und um 1 Bit nach rechts geschiftet, damit der Wert halbiert
    wird und der Zielvektor spaeter keinen Ueberlauf hat.
    -- Um wieder die vollen 13 Bit zu erhalten, wird die resize-
    Funktion verwendet.
390
    -- Real
392
    temp14bit_re := resize(mult_re(bit_width_multiplier-4 downto
bit_width_multiplier-4-bit_width_extern), bit_width_adder+1) + resize(
temp_re(0)(bit_width_adder-1 downto 1), bit_width_adder+1);
    temp_re(0) := temp14bit_re(bit_width_adder downto 1);

396
    -- Imag
398    temp14bit_im := resize(mult_im(bit_width_multiplier-4 downto
bit_width_multiplier-4-bit_width_extern), bit_width_adder+1) + resize(
temp_im(0)(bit_width_adder-1 downto 1), bit_width_adder+1);
    temp_im(0) := temp14bit_im(bit_width_adder downto 1);

```

```

400      — Indizes vertauschen -> Transponiert abspeichern
402      if dft_1d_2d = '0' then
403          dft_1d_real(l_col)(row_col_idx) := temp_re(0)(bit_width_adder-1
404      downto 1);
405          dft_1d_imag(l_col)(row_col_idx) := temp_im(0)(bit_width_adder-1
406      downto 1);
407      else
408          result_real(l_col)(row_col_idx) <= temp_re(0)(bit_width_adder-1
409      downto 1);
410          result_imag(l_col)(row_col_idx) <= temp_im(0)(bit_width_adder-1
411      downto 1);
412      end if;
413
414      next_dft_state <= twiddle_calc;
415      if element = 63 then
416          if dft_1d_2d = '1' then
417              next_dft_state <= set_ready_bit;
418          end if;
419          dft_1d_2d := not dft_1d_2d;
420          dft_1d_2d_out <= dft_1d_2d;
421      end if;
422
423      element := element+1;
424      element_out <= element;
425
426      when set_ready_bit =>
427          result_ready <= '1';
428          next_dft_state <= twiddle_calc;
429
430      when others => next_dft_state <= twiddle_calc;
431      end case;
432  end process;
433  end arch;

```

Listing C.7: Berechnung der 2D-DFT

```

library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4 library work;
5 use work.all;
6 use constants.all;
7 use datatypes.all;
8
9 entity dft8optimiert_top is
10 —     port(
11 —         result_real : out t_2d_array;

```

```

12  —         result_imag : out t_2d_array
13  —         );
14  end entity dft8optimiert_top;

16  architecture arch of dft8optimiert_top is

18      signal nReset      : bit;
19      signal clk          : bit;
20      signal input_real   : t_2d_array;
21      signal input_imag   : t_2d_array;
22      signal result_real  : t_2d_array;
23      signal result_imag  : t_2d_array;
24      signal loaded       : bit;
25      signal result_ready : bit;
26      signal write_done   : bit;
27      signal idft         : bit := '0';

28
29      signal state_out     : t_dft8_states;
30      signal element_out   : unsigned(5 downto 0);
31      signal dft_1d_2d_out : bit;
32
33
34  component dft8optimiert
35      port(
36          clk          : in  bit;
37          nReset       : in  bit;
38          loaded       : in  bit;
39          input_real    : in  t_2d_array;
40          input_imag    : in  t_2d_array;
41          result_real   : out t_2d_array;
42          result_imag   : out t_2d_array;
43          result_ready  : out bit;
44          idft         : in  bit;
45          state_out     : out t_dft8_states;
46          element_out   : out unsigned(5 downto 0);
47          dft_1d_2d_out : out bit
48      );
49  end component;

50
51
52  component read_input_matrix
53      port(
54          clk          : in  bit;
55          loaded       : out bit;
56          input_real   : out t_2d_array;
57          input_imag   : out t_2d_array
58      );
59  end component;

60
61
62  component write_results

```

```

64     port(
65         result_ready : in bit;
66         result_real  : in t_2d_array;
67         result_imag  : in t_2d_array;
68         write_done   : out bit
69     );
70 end component;

72 begin
73     dft : dft8optimiert
74         port map(
75             nReset      => nReset ,
76             clk          => clk ,
77             loaded       => loaded ,
78             input_real   => input_real ,
79             input_imag   => input_imag ,
80             result_real  => result_real ,
81             result_imag  => result_imag ,
82             result_ready => result_ready ,
83             idft         => idft ,
84             state_out    => state_out ,
85             element_out  => element_out ,
86             dft_1d_2d_out => dft_1d_2d_out
87         );
88
89     mat : read_input_matrix
90         port map(
91             clk          => clk ,
92             loaded       => loaded ,
93             input_real   => input_real ,
94             input_imag   => input_imag
95         );
96
97     write : write_results
98         port map(
99             result_ready => result_ready ,
100             result_real  => result_real ,
101             result_imag  => result_imag ,
102             write_done   => write_done
103         );
104
105     clk    <= not clk after 20 ns;
106     nReset <= '1' after 40 ns;
107
108 end arch;

```

Listing C.8: Top-Level-Entität der 2D-DFT

D. Testumgebung

D.1. Bash-Skript zum Ausführen des Tests

```
#!/bin/bash
2
matlab_script="VHDL-DFT_Vergleich.m"
4
./simulate.sh && matlab -nojvm -nodisplay -nosplash -r $matlab_script
6
stty echo
```

Listing D.1: Aufruf der Testumgebung, Vergleich von VHDL- und Matlab-Ergebnissen

D.2. Skript zum Kompilieren und Simulieren des VHDL-Programms

```
1 #!/bin/bash
3 # global settings
5 errormax=15
worklib=worklib
7 #testbench=top_level_tb
testbench=dft8optimiert_top
9 architecture=arch
simulation_time="1500ns"
11
13 # VHDL-files
15 constant_declarations="constants.vhdl"
datatype_declarations="datatypes.vhdl"
17
main_entity="dft8optimiert.vhdl"
19 top_level_entity="dft8_optimiert_top.vhdl"
#top_level_testbench=
21
embedded_entity_1="read_input_matrix.vhdl"
23 embedded_entity_2="write_results.vhdl"
25
```



```
constant_declarations=$directory$constant_declarations
27 datatype_declarations=$directory$datatype_declarations
function_declarations=$directory$function_declarations
29 main_entity=$directory$main_entity
top_level_entity=$directory$top_level_entity
31 #top_level_testbench=$directory$top_level_testbench

33 embedded_entity_1=$directory$embedded_entity_1
embedded_entity_2=$directory$embedded_entity_2
35

37 # libs und logfiles

39 cdslib="cds.lib"
elab_logfile="ncelab.log"
41 ncvhdl_logfile="nchvdl.log"
ncsim_logfile="ncsim.log"
43
cdslib=${base_dir}${work_dir}${cdslib}
45 elab_logfile=${directory}${elab_logfile}
ncvhdl_logfile=${directory}${ncvhdl_logfile}
47 ncsim_logfile=${directory}${ncsim_logfile}
49
##
51
ncvhdl \
53 -work $worklib \
-cdslib $cdslib \
55 -logfile $ncvhdl_logfile \
-errormax $errormax \
57 -update \
-v93 \
59 -linedebug \
$constant_declarations \
61 $datatype_declarations \
$embedded_entity_1 \
63 $embedded_entity_2 \
$main_entity \
65 $top_level_entity \
#$top_level_testbench
67 #-status \

69 ncelab \
-work $worklib \
71 -cdslib $cdslib \
-logfile $elab_logfile \
73 -errormax $errormax \
-access +wc \
75 ${worklib}.${testbench}
#-status \
```

```

77 ncsim \
79 -cdslib $cdslib \
  -logfile $ncsim_logfile \
81 -errormax $errormax \
  -exit \
83 ${worklib}.${testbench}:${architecture} \
  -input simulationTime.tcl
85 #-status \

87

89 #ncvhdl -work worklib -cdslib /home/tlattmann/cadence/mat_mult/cds.lib -
  logfile /home/tlattmann/cadence/mat_mult/nchvdl.log -errormax 15 -update
  -v93 -linedebug /home/tlattmann/cadence/mat_mult/HDL/constants.vhdl /
  home/tlattmann/cadence/mat_mult/HDL/datatypes.vhdl /home/tlattmann/
  cadence/mat_mult/HDL/functions.vhdl /home/tlattmann/cadence/mat_mult/HDL
  /read_input_matrix.vhdl /home/tlattmann/cadence/mat_mult/HDL/
  write_results.vhdl /home/tlattmann/cadence/mat_mult/HDL/dft8optimiert.
  vhdl /home/tlattmann/cadence/mat_mult/HDL/dft8_optimiert_top.vhdl -
  status

91 #ncelab -work worklib -cdslib /home/tlattmann/cadence/mat_mult/cds.lib -
  logfile /home/tlattmann/cadence/mat_mult/ncelab.log -errormax 15 -access
  +wc worklib.dft8optimiert_top -status

93 #ncsim -cdslib /home/tlattmann/cadence/mat_mult/cds.lib -logfile /home/
  tlattmann/cadence/mat_mult/ncsim.log -errormax 15 worklib.
  dft8_optimiert_top:arch -input simulationTime.tcl -status

95 #database -open waves -into waves.shm -default
#probe -create -shm :clk :input_imag :input_real :loaded :mult_im_out :
  mult_re_out :multState_out :nReset :result_imag :result_ready :
  result_real :sum1_stage1_3v6_re_out :sum1_stage2_2v3_re_out :
  sum1_stage2_3v3_re_out :sum1_stage3_1v1_re_out :sum3_stage1_im_out :
  sum3_stage1_re_out :sum3_stage2_im_out :sum3_stage2_re_out :
  sum3_stage3_im_out :sum3_stage3_re_out :sum3_stage4_im_out :
  sum3_stage4_re_out :write_done

```

Listing D.2: Simulations des VHDL-Quelltextes

```
run 32us
```

Listing D.3: Dauer der Simulation

D.3. Matlab-Skript zur Berechnung der Referenzwerte und Vergleich der Ergebnisse

```

1 filename_2 = 'InputMatrix_komplex.txt';
  filename_1 = 'Results.txt';

3
  delimiterIn = ' ';
5 bit_width_extern = 12

7 Input_bin = importdata(filename_2, delimiterIn);
  Input_bin_real = Input_bin(:,1:2:end);
9 Input_bin_imag = Input_bin(:,2:2:end);

11 Results_vhdl_bin = importdata(filename_1, delimiterIn);
  Results_vhdl_bin_real = Results_vhdl_bin(:,1:2:end);
13 Results_vhdl_bin_imag = Results_vhdl_bin(:,2:2:end);

15 Input_dec_imag = nan(8);
  Results_vhdl_dec_real = nan(8);
17 Results_vhdl_dec_imag = nan(8);
  Result_octave_real_1d = nan(8);
19 Result_octave_imag_1d = nan(8);

21 a=fi(0,1,bit_width_extern,bit_width_extern-2);

23 N = 8;
  for m = 1:N
25     for n = 1:N
          a_bin=mat2str(Results_vhdl_bin_real(m,n),bit_width_extern);
27         Results_vhdl_dec_real(m,n) = a_bin.double;
          a_bin=mat2str(Results_vhdl_bin_imag(m,n),bit_width_extern);
29         Results_vhdl_dec_imag(m,n) = a_bin.double;

          a_bin=mat2str(Input_bin_real(m,n),bit_width_extern);
31         Input_dec_real(m,n) = a_bin.double;
          a_bin=mat2str(Input_bin_imag(m,n),bit_width_extern);
33         Input_dec_imag(m,n) = a_bin.double;
35     end
  end

37

39 Input_dec=Input_dec_real+1i*Input_dec_imag;
  TW=exp(-i*2*pi*[0:7]'*[0:7]/8);
41

  %Result_octave_1d=TW*Input_dec;
43 %Result_octave_real_1d=real(Result_octave_1d)/16
  %Result_octave_imag_1d=imag(Result_octave_1d)

45
  Result_octave=TW*Input_dec*TW.';
47 Result_octave=Result_octave./256;

49 Results_vhdl_dec_real
  Result_octave_real=real(Result_octave)
51

```

```
Result_octave_imag=imag(Result_octave);  
53 Results_vhdl_dec_imag;  
  
55 diff_real=Result_octave_real-Results_vhdl_dec_real  
diff_imag=Result_octave_imag-Results_vhdl_dec_imag;  
57  
quit
```

Listing D.4: Berechnung der Differenzen der DFT in Matlab und VHDL

E. Cadence Tutorial

Encounter-Tutorial

Thomas Lattmann

18. April 2018

Inhaltsverzeichnis

1	Einleitung	3
1.1	Überblick	3
1.2	Was ist Cadence?	3
1.3	Server-Login	3
1.3.1	Der Bash-Befehl	3
1.3.2	Einstellungen in der SSH-Konfiguration speichern	4
1.3.3	Bash-Befehl zum Einbinden des Serverdateisystems	4
1.3.4	Login- & Mount-Skript für den Cadence-Server	4
2	Vorbereitungen	6
3	Durchführung	7
3.1	HDL-Synthese mit <code>rc</code>	7
3.2	Von der Netzliste zum Core	8
3.2.1	ams-spezifische Befehle	8
3.2.2	Design-Konfigurationsdatei für <code>encounter</code> erstellen	8
3.3	<code>encounter</code>	10
4	Anmerkungen	14
4.1	Tabelle mit Erläuterungen	14
4.2	Tabelle mit Dateiendungen	14
4.3	Tabelle mit den gängigen Linux-Bashbefehlen	15
4.4	Shell-Umgebungsvariablen beim Login laden	15
4.5	Hilfe zu <code>find</code>	16
4.6	Dateien editieren	16
5	Anhang	17
5.1	<code>rc_RTL.tcl</code> - Skript zur HDL-Synthetisierung	17
5.2	<code>counter.vhdl</code>	19
5.3	<code>Testbench</code>	20
5.4	<code>counter.v</code> (verilog-Code)	21
5.5	<code>amsSetup.tcl</code> - zusätzliche ams-Befehle für <code>encounter</code>	21
5.6	<code>c35b3.conf</code> - Design-Konfigurationsdatei	29
5.7	Skript für einfachen Login	31

1 Einleitung

1.1 Überblick

Dieses Tutorial soll ein detaillierter Leitfaden für den Einstieg in die Cadence-Umgebung sein. Es wird das Design eines vollständigen Chip-Cores inklusive Timingsimulation mit den Standardzellen der Firma Austria Microsystems (ams) behandelt. In diesem Tutorial wird die Bibliothek `c35b3` verwendet, welche festlegt, dass es sich um einen $35\mu\text{m}$ -Prozess mit 3 Metalllagen handelt. Die Herangehensweise wird beispielhaft an einem einfachen Zähler erläutert, welcher in Listing 5.2 zu finden ist. Listing 5.3 ist die zur Simulation in Modelsim oder NCSim nötige Testbench. Die Anleitung beschränkt sich i.d.R. auf die Befehle für die Kommandozeile. In Abschnitt 1.3.3 wird beschrieben, wie sich das Dateisystem des Servers lokal einbinden lässt. Auf diese Weise kann der lokale, grafische Dateimanager genutzt und aus ihm heraus zum Beispiel Textdateien geöffnet werden.

1.2 Was ist Cadence?

Es handelt sich bei Cadence um eine Sammlung von Programmen mit denen es möglich ist Design und Simulation eines Cores durchzuführen. Cadence alleine bringt jedoch „nur“ Tools mit, mit denen man selbst entwickelte Zellen (Gatter, Leitungen, Pads, ...) in Software beschrieben und in einer Bibliothek abspeichern kann, um diese dann später für die Entwicklung des Cores (bzw. dessen „Bauplan“) zu verwenden. Da dieser Ansatz sehr Umfangreich ist, stellen viele Firmen die von ihnen entwickelten sog. Standardzellen bereit, um sie in Cadence zu integrieren. So ähnlich wie Toolboxen in Matlab, nur das sie von anderen Firmen vertrieben werden. An der HAW Hamburg wird die „Toolbox“ von `ams` verwendet, welche neben den Bibliotheken auch eigene Programme mitbringt. Teilweise ist es auch so, dass mit `ams`-Befehlen Cadence-Programme gestartet werden, jedoch direkt `ams`-spezifische Einstellungen vorgenommen werden. Die Verwendung der Standardzellen von `ams` bedingt, dass der Chip auch von dort hergestellt wird.

1.3 Server-Login

1.3.1 Der Bash-Befehl

Der Login auf dem Server erfolgt aus der Kommandozeile (z.B. **Terminal**) mittels

```
$ ssh -X <username>@141.22.14.104 -i /.ssh/<id_rsa-Datei>
```

wobei das `-X` bewirkt, dass als Display der lokale Monitor gesetzt wird, sodass grafische

Programme gestartet werden können und auf dem eigenen Monitor angezeigt werden. Mit `-i` kann eine bestimmte `id_rsa`-Datei gewählt werden. Dies ist dann wichtig, wenn es mehrere gibt bzw. die zu verwendende nicht den Standardname `id_rsa` hat. Wenn auf dem lokalen Rechner und dem Server die selben Benutzernamen verwendet werden, kann `<user>` weggelassen werden.

1.3.2 Einstellungen in der SSH-Konfiguration speichern

Alternativ können die Verbindungseinstellungen auch in der Datei `~/.ssh/config` gespeichert werden.

```
$ nano ~/.ssh/config
```

```
host                cadence
hostname            141.22.14.104
user                <user>
identityfile        /home/<user>/.ssh/<id_rsa-Datei>
forwardX11          yes
```

Nun kann der Login über ein einfaches

```
$ ssh cadence
```

erfolgen, wobei der Name `cadence` auch anders gewählt werden kann.

1.3.3 Bash-Befehl zum Einbinden des Serverdateisystems

Um unabhängig von dem Skript das Dateisystem des Servers in ein lokales Verzeichnis einzubinden, muss zunächst ein Ordner erstellt werden. Anschließend kann mit `sshfs <user>@141.22.14.104:/home <Pfad/zum/Ordner> -o IdentityFile=id_rsa-Datei` eingebunden werden.

1.3.4 Login- & Mount-Skript für den Cadence-Server

Als weitere Alternative kann das Skript `run_cadence` (5.7) verwendet werden, welches auf privaten Rechnern versucht den richtigen Benutzernamen für den Server zu erraten. Voraussetzung ist, dass auf dem eigenen Rechner Vor- und Nachname angegeben wurden und auf dem Server der Benutzername im Format Anfangsbuchstabe des Vornamens gefolgt vom Nachnamen ist. Bei den Laborrechnern wird davon ausgegangen, dass der gleiche Benutzername vergeben wurde, wie auf dem Server.

Zusätzlich zum Login wird das Verzeichnis `/home` des Server auf dem eigenen Rechner mittels `sshfs` unter `/home/<cadence_server>` eingebunden. Auf diese Weise lassen sich die Verzeichnisse im Dateimanager durchsuchen und die Dateien mit den lokal installierten Editoren bearbeiten. Wenn das Verzeichnis nicht vorhanden ist, wird es erstellt. Dadurch, dass das Passwort für die `id_rsa`-Datei von `ssh-agent` gespeichert wird, ist es bis zum Neustart des Rechners bei weiteren Server-Logins nicht nötig dieses erneut einzugeben.

Wenn nun noch die `run_cadence.desktop` aus Listing 5.8 in das Verzeichnis `/home/<user>/.local/share/applications/` kopiert wird, lässt sich der Login komfortabel aus dem Startmenü heraus starten. Die `.desktop`-Datei erwartet ein `cadence_icon.jpg` im selben Verzeichnis und geht davon aus, dass das Skript in `/usr/bin/` liegt. Letzteres ermöglicht es einem auch aus jedem beliebigen Ordner in der Kommandozeile das Skript auszuführen.

2 Vorbereitungen

1. Projektverzeichnis erstellen

```
$ mkdir <Verzeichnis>
```

2. in Projektverzeichnis wechseln

```
$ cd <Verzeichnis>
```

3. VHDL-Verzeichnis erstellen

```
$ mkdir HDL
```

4. VHDL-Code in das neue Verzeichnis kopieren

```
$ cp <Quelle>/counter.vhdl HDL
```

5. von Cadence benötigte Shell-Umgebungsvariablen setzen. Die Variablen sind in dem Bash-Skript `cadence_env_new.sh` zusammen gefasst.

```
$ source /home/cdsmgr/cadence_env_new.sh
```

6. die zu verwendende Technologie auswählen. Hier kann zwischen verschiedenen Prozess-Größen sowie Anzahl Metalllagen gewählt werden. Für einen $35\mu\text{m}$ -Prozess mit drei Metalllagen ist folgender Befehl einzugeben:

```
$ cd .. $ ams_cds -tech c35b3 &
```

Wenn ein neues Projekt angelegt wird, erstellt der Befehl beim ersten Aufruf mehrere Dateien, öffnet aber noch keine Fenster. In diesem Fall muss der Befehl ein zweites Mal ausgeführt werden. Zumindest der Virtuoso Log und der Library Manager sollten nicht geschlossen werden, sie werden ggf. im weiteren Verlauf noch benötigt.

7. im Fenster „Select Process Option“ den Prozess **C35B3C3 PIP VG5 HIRES** auswählen.

Im „Virtuoso Log“ ist zu sehen, dass in die `cds.lib` im Projektverzeichnis der Pfad zur ausgewählten Prozess-Bibliothek eingetragen wird. Die Datei `cds.lib.save` ist abgesehen von dieser Ergänzung identisch. Die Programme, die im weiteren Verlauf verwendet werden, überschreiben die `cds.lib`. Neben diesen Dateien werden noch etliche weitere erstellt.

3 Durchführung

3.1 HDL-Synthese mit `rc`

Der auf RTL-Ebene geschriebene VHDL-Code (5.2) wird mit `rc` in eine Verilog-Gate-Level-Netzliste (Verilog-Code, 5.4) umgewandelt. Um nicht die einzelnen Schritte durchführen zu müssen, startet man das Programm sinnvoller Weise mit dem TCL-Skript aus Listing 5.1, welches noch entsprechend angepasst werden muss.

Da im Hintergrund einige Dateien erzeugt werden, lohnt es sich ein eigenes Verzeichnis für `rc` zu erstellen und das Programm aus diesem heraus zu starten.

```
8. $ mkdir rc
9. $ cd rc
10. $ cp <Quellverzeichnis>/rc_RTL.tcl .
11. $ nano rc_RTL.tcl
12. $ rc -f rc_RTL.tcl -gui
13. rc:/> exit
14. $ cd ..
```

Das Ergebnis ist unter anderem der synthetisierte Verilog-Code, welcher im nächsten Schritt benötigt wird. Hier wird auch zum ersten Mal eine `.sdf` Datei erzeugt, welche die Informationen für die Laufzeiten der Gatter und Leitungen beinhaltet (standard delay format). Da aber zu diesem Zeitpunkt keine Informationen über die verwendeten Gatter und Leitungslängen vorliegen, haben alle Verzögerungen den Wert 0. Wenn die Einstellungen aus der `rc_RTL.tcl` übernommen wurden, liegen die genannten Dateien in `<Projektverzeichnis>/VERILOG`. Dies wurde so gewählt, weil das Programm im nächsten Schritt davon ausgeht, dass sich die Verilog-Datei dort befindet.

Wenn das Programm ohne die Option `-gui` aufgerufen wurde oder die grafische Oberfläche mit `File → Hide GUI` geschlossen wurde, lässt sie sich über das Kommando `gui_show` aufrufen. In der GUI wird die Netzliste graphisch dargestellt, mehr ist jetzt noch nicht möglich.

Für einen besseren Workflow kann beim Aufruf von `rc` das `-gui` weggelassen und das Kommentarzeichen in der letzten Zeile der `rc_RTL.tcl` vor `quit` entfernt werden. Nun werden die benötigten Dateien erstellt, anschließend beendet sich das Programm.

3.2 Von der Netzliste zum Core

Bevor `encounter` genutzt werden kann, müssen noch die Dateien `amsSetup.tcl` sowie eine `c35b3_*.conf` erzeugt werden. Die erstgenannte wird mit `ams_encounter` erstellt, die zweite mit `ams_edi`. Das Vorgehen wird nachfolgend beschrieben.

3.2.1 ams-spezifische Befehle

Damit in `encounter` die nötigen Befehle verfügbar sind, muss zunächst ein Skript mit den entsprechenden Designeinstellungen erzeugt werden.

```
15. $ ams_encounter -tech c35b3
```

Später wird dieses Skript in `encounter` ausgeführt.

3.2.2 Design-Konfigurationsdatei für `encounter` erstellen

Es gibt drei verschiedene Möglichkeiten die `c35b3_*.conf` zu erstellen, welche nachfolgend erklärt werden. Zu beachten ist, dass in allen Fällen davon ausgegangen wird, dass die Verilog-Dateien im Unterverzeichnis `VERILOG` liegen. Entsprechend wurde in der `rcRTL.tcl` dieser Ordner als Pfad für die neu erzeugten Verilog-Dateien angegeben.

Da standardmäßig verschiedene Spannungsversorgungen und Massen auf dem Core vorgesehen sind, in diesem Design aber nur `vdd!` und `gnd!` Verwendung finden, sollten die anderen entweder vor Aufruf von `encounter` in der `amsSetup.tcl` (Listing 5.5) in den Zeilen 89 - 94 oder später in der grafischen Oberfläche von `encounter` entfernt werden.

Methode 1: Grafisches Erstellen mit `ams_edi`

Das Programm wird mit

```
16. $ ams_edi &
```

aufgerufen.

Die Einstellungen in Abbildung 3.1 müssen übernommen werden. Mit Klick auf **Create Setup Files** wird u.a. eine `c35b3_*.conf` erstellt, wobei das `*` für die angegebene Top-Level Cell steht. Also beispielsweise `counter`. Zusätzlich wird bei dieser Methode die ausführbare Det `amsEdiSetup` angelegt.

Methode 2: Erstellen mit `ams_edis`, geeignet für Skripte

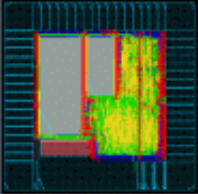
Alternativ kann das Programm `ams_edis` mit allen Einstellungen direkt in der Konsole die benötigten Dateien erstellen. Dies ist vor allem für die Verwendung in Skripten interessant.

✕

⏮

⏭

Set Encounter Environment



amU

hitkit

EDI Place & Route Setup

Hit-Kit Version

/home/cdsmgr/ams

Encounter Version

EDI10.1USR2

Cadence Version

Cadence6-OA

Verilog Netlist Name

counter_syn.v

Verilog TopCell Name

counter

Technology

c35

Metal Layers

3M

Corelib

CORELIB

Core-Timing-Lib

Best

_3.3V + 10% => 3.63V

Typ

_3.3V

Worst

_3.3V - 10% => 2.9699999999999998V

IOlib

IOLIB

IO-Timing-Lib

Best

_3.3V + 10% => 3.63V

Typ

_3.3V

Worst

_3.3V - 10% => 2.9699999999999998V

Create Source File

Create Setup Files

Close

```
$ ams_edis -tech c35 -metlay 3M -vn counter_syn.v -vt counter  
-corelib CORELIB -corevolt _3.3V -corevolt_bc _3.3V -corevolt_wc _3.3V  
-iolib IOLIB -iovolt _3.3V -iovolt_bc _3.3V -iovolt_wc _3.3V
```

Methode 3: Das `amsEdiSetup`-Skript verwenden

Diese Datei existiert erst nachdem die grafische Methode (`ams_edi`) einmal ausgeführt wurde. Mit ihr ist es möglich die Dateien mit den selben Einstellungen wie eingangs gewählt erneut zu erstellen.

```
$ ./amsEdiSetup
```

3.3 encounter

Nach dem die Dateien auf eine der beschriebenen Weisen erstellt wurden, kann `encounter` gestartet werden (im Vordergrund, also ohne `&`). Anschließend werden die in der `amsSetup.tcl` gesetzten Variablen geladen.

```
17. $ encounter
```

```
18. encounter 1> source amsSetup.tcl
```

Jetzt stehen in der Shell die `encounter` bereit stellt weitere, `ams`-spezifische Befehle zur Verfügung. Sie fangen alle mit `ams` an, erwähnenswert ist an dieser Stelle `amsHelp`, welcher eine Übersicht und Kurzschreibung anzeigt.

Alle Befehle, egal ob in der Kommandozeile eingegeben oder über Menüs getätigt, finden sich neben zurückgemeldeten Informationen sowohl in der Datei `encounter.log` als auch in der `encounter.cmd`. Um nur die Befehle zu sehen, die hinter den Menüeingaben stehen, kann man folgenden Befehl in einem zweiten Terminalfenster eingeben:

```
$ tail -f encounter.log | grep "<CMD>"
```

`encounter.cmd` erwähnen!

Im nächsten Schritt wird die Design-Konfigurationsdatei geladen:

```
19. File → Import Design → Load → c35b3_counter.conf
```

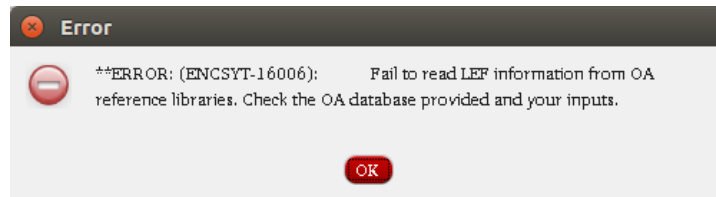
Es müssen ggf. noch im Abschnitt **Power** bei **Power Nets** und **Ground Nets** die nicht benötigten Netze entfernt werden, so dass nur noch `vdd!` und `gnd!` dort eingetragen sind.

Alternativ kann das Design auch mit

```
encounter 2> source c35_counter.conf  
encounter 3> init_design  
encounter 4> fit
```

geladen und auf die Fenstergröße angepasst werden. In diesem Fall müssen zuvor die oben genannten nicht benötigten `gnd_x!` und `vdd_x!` aus der Konfigurationsdatei entfernt werden.

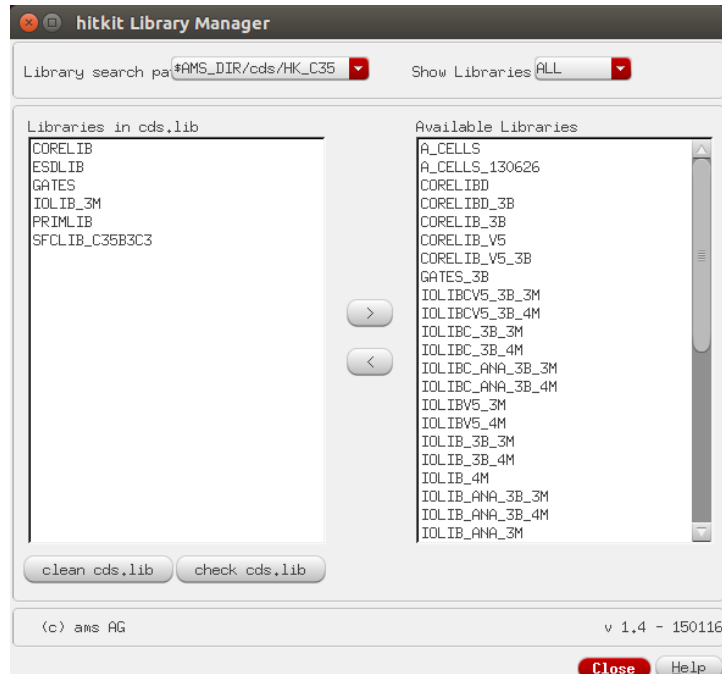
Falls es beim Laden zu der Fehlermeldung `ERROR: Failed to read LEF information from OA reference Library.` kommt, sollte zunächst encounter vollständig geschlossen werden.



Anschließend muss wie folgt vorgegangen werden:

Virtuoso Log → hitkit → Add/Remove hitkit libraries

Im hitkit Library Manager müssen den Bibliotheken, welche in der `cds.lib` eingetragen sind, die `CORELIBD` entfernt und die `CORELIB` sowie `IOLIB_3M` hinzugefügt werden.



Im Menü Floorplan → Specify Floorplan kann die Größe des Cores angegeben werden.



20. encounter 5> amsGlobalConnect core

21. encounter 6> amsAddEndCaps

22. Power → Power Planning → Add Ring

23. Power → Power Planning → Add Stripes

24. Route → Special Route

Add Rings

Basic Advanced Via Generation

Net(s): gnd! vdd! ...

Ring Type

☒ Core ring(s) contouring

☒ Around core boundary ☐ Along I/O boundary

☐ Exclude selected objects

☐ Block ring(s) around

☐ Each block

☐ Each reef

☐ Selected power domain/fences/reefs

☐ Each selected block and/or group of core rows

☐ Clusters of selected blocks and/or groups of core rows

☐ With shared ring edges

☐ User defined coordinates: MouseClick

☐ Core ring ☐ Block ring

Ring Configuration

	Top:	Bottom:	Left:	Right:
Layer:	MET3 H	MET3 H	MET2 V	MET2 V
Width:	1	1	1	1
Spacing:	0.6	0.6	0.6	0.6
Offset:	<input type="radio"/> Center in channel	<input checked="" type="radio"/> Specify		
	0.7	0.7	0.7	0.7

Option Set

☐ Edit Add Ring Option

OK Set Mode Apply Defaults Cancel Help

Add Stripes

Basic Advanced Via Generation

Set Configuration

Net(s): gnd! vdd! ...

Layer: MET2 Direction: ☒ Vertical ☐ Horizontal

Width: 1 Spacing: 0.6 Update

Set Pattern

☐ Set-to-set distance: 100

☒ Number of sets: 1

☐ Bumps ☐ Over ☐ Between

☐ Over P/G pins Pin layer: Top pin layer Max pin width: 0

☐ Master name: ☐ Selected blocks ☐ All blocks

Stripe Boundary

☒ Core ring

☐ Pad ring ☐ Inner ☐ Outer

☐ Design boundary ☒ Create pins

☐ Each selected block/domain/fence

☐ All domains

☐ Specify rectangular area

☐ Specify rectilinear area

First/Last Stripe

Start from: ☒ left ☐ right

☒ Relative from core or selected area

X from left: 27 X from right: 0

☐ Absolute locations

Option Set

☐ Edit Add Stripe Option

OK Set Mode Apply Defaults Cancel Help

1. amsFillcore

4 Anmerkungen

4.1 Tabelle mit Erläuterungen

RTL	Register-Transfer-Level (-Ebene)
Gate-Level	Gatterebene
OA	Open Access, alternatives Layout-Speicherformat zu LEF-Bibliotheken, binär
MMMC	Multi Mode Multi Corner
PnR	Place and Route
CTS	Clock Tree Synthesis

4.2 Tabelle mit Dateieindugen

vhdl	Hardwarebeschreibungssprache, muss für Cadence nach verilog konvertiert werden. z.B. mit rc
v	verilog, die von der Cadence -Umgebung verwendete HDL
tcl	Skript-Sprache, wird z.B. zum Laden von Umgebungsvariablen in der encounter -Shell verwendet
conf	Konfigurationsdatei zum Speichern bzw. Laden von Einstellungen (ams_edi bzw. encounter)
sdf	standard delay format, Verzögerungen der Gatter und Leitungen
sdf.X	kompilierte sdf-Datei
sdc	standard delay/design constraints?
lib	Bibliotheken für die Gatter
lef	Layout Exchange Format
oa	Open Exchange, Nachfolger von LEF
def	design exchange format?
ref	
io	IO assignment
clocktch	Clock specifications
captab	Capacitance table
map	Stram-out layer map
view	MMMC-View

4.3 Tabelle mit den gängigen Linux-Bashbefehlen

bash	Kommandozeile (Terminal) [Bourne Again Shell]
ssh	secure shell (sicheres Telnet)
\$	Prompt der Bash
#	Prompt der Bash als root (Administrator)
sudo <Befehl>	Befehl mit root-Rechten ausführen, sofern man diese hat
.	aktuelles Verzeichnis
..	eine Verzeichnisebene höher
/	Wurzelverzeichnis bzw. Trenner zwischen Verzeichnisebenen
<Befehl> &	bewirkt, dass die Eingabe in der Shell im Hintergrund ausgeführt wird.
<user>	der eigene Benutzername, dieser kann auf dem lokalen Rechner ein anderer sein, als auf dem Server
~	synonym mit /home/<user> (in der Kommandozeile)
cd <Verzeichnis>	change directory
cd	wechselt in das Home-Verzeichnis
cp <Quelle> <Ziel>	copy
mv <Quelle> <Ziel>	move / rename
rm <Datei>	remove
rm -r	rekursive remove, Unterverzeichnisse mit einbeziehen
rm -f	force remove, schreibgeschützte Dateien löschen
rm *	Inhalt des Verzeichnisses löschen
ls	list, Verzeichnisinhalt anzeigen
ls -l	list long, ausführlich Informationen anzeigen
ls -a	list all, auch versteckte Dateien anzeigen
.<dateiname>	versteckte Datei

4.4 Shell-Umgebungsvariablen beim Login laden

Um sich einen Arbeitsschritt zu ersparen, kann man einmalig den entsprechenden Befehl in die versteckte Datei `.bashrc` im eigenen Home-Verzeichnis eintragen. Befehle in dieser Datei werden bei jedem Login automatisch ausgeführt.

```
$ cd
$ nano .bashrc

if [ -f /home/cdsmgr/cadence_env_new.sh ]; then
    source /home/cdsmgr/cadence_env_new.sh && echo "Cadence-Umgebungsvariablen
geladen"
else
```

```
    echo "Cadence-Umgebungsvariablen konnten nicht geladen werden"
fi
```

4.5 Hilfe zu find

Beispiel: Suche nach Dateien in <Verzeichnis> und allen Unterverzeichnissen mit genau dem Namen `dateiname`, inklusive der Berücksichtigung von Groß- und Kleinschreibung.

```
$ find <Verzeichnis> -type f -name "dateiname"
```

Häufige Optionen für `find` finden sich in der nachfolgenden Tabelle:

<code>-name "Datei.txt"</code>	genauer Dateiname, Groß-/Kleinschreibweise wird beachtet
<code>-iname "datei.txt"</code>	genauer Dateiname, Groß-/Kleinschreibweise wird nicht beachtet
<code>!-name "Datei"</code>	"Datei" kommt nicht im Namen vor
<code>datei.*</code>	sucht nach Dateien mit gleichem Namen aber unterschiedlicher Endung
<code>*datei</code>	sucht nach Dateien mit gleichem Namensende.
<code>*datei*</code>	datei muss irgendwo im Namen vorkommen
<code>*</code>	beliebige Datei
<code>-type f</code>	es werden nur Dateien gesucht
<code>-type d</code>	es werden nur Verzeichnisse gesucht

Die Anführungszeichen sind nur bei Namen nötig, die Zeichen enthalten, die von der Bash interpretiert werden würden, bevor sie von `find` verwendet werden.

Wenn `find` auf Unterverzeichnisse stößt, für die man keine Berechtigung hat, erscheint eine Fehlermeldung. Da diese das Suchergebnis oft unübersichtlich machen, kann es hilfreich sein, die Standard-Fehlerausgabe nach `/dev/null` umzulenken. Um die Suche im aktuellen Verzeichnis zu beginnen und Fehlermeldungen nicht angezeigt zu bekommen, verwendet man `find` folgendermaßen:

```
$ find . -name name 2>/dev/null
```

4.6 Dateien editieren

1) In der Konsole:

```
$ nano Datei
```

2) Graphischen Editor des Servers auf dem eigenen Bildschirm anzeigen:

```
$ gedit Datei &
```

3) Wenn das Dateisystem des Servers wie in 1.3.3 beschrieben eingebunden wurde, kann die Datei aus dem Dateimanager heraus geöffnet werden.

5 Anhang

5.1 rc_RTL.tcl - Skript zur HDL-Synthethisierung

```
1 #####
2 ## Title      : Encounter(R) RTL Compiler – Script
3 ## Project    : ESZ–ABS
4 #####
5 ## File       : vhdlRTL.tcl
6 ## Author     : Daniel Sabotta
7 ## Company    : HAW Hamburg
8 ## Created    : 2012–05–23
9 ## Last update: 2012–05–23
10 #####
11 ## Description: Load this file into Encounter(R) RTL Compiler
12 ##              – Script to synthesize a vhdl file
13 ##
14 ##              Usage: "rc -f vhdlRTL.tcl"
15 ##
16 ##              Original file from Erik Brunvand, 2008
17 #####
18 ## Revisions  :
19 ## Date        Version  Author  Description
20 #####
21
22 ## Set the search paths to the libraries and HDL
23 ## Remember that "." yout current directory
24
25 ## Search Path for VHDL and Verilog files
26 set_attribute hdl_search_path { ../HDL };
27 ## Search Path for library files
28 set_attribute lib_search_path { /home/cdsmgr/ams/liberty/c35_3.3V };
29 ## Target library
30 set_attribute library [list c35_CORELIB_TYP.lib];
31 ## See a lot of warnings
32 set_attribute information_level 6;
33
34
35 set hdlFiles [list counter.vhdl] ;# All HDL files
36 set basename counter ;# name of toplevel module
```

```

37 set savePath      {./}    ;# Save created Files here
38 set runName       syn      ;# name appended to output files
39
40 set clkName        clk      ;# clock name
41 set clkPeriod_ps   62500    ;# clock periode in ps
42 set clkInDelay_ps  250      ;# delay from clock to input valid
43 set clkOutDelay_ps 250      ;# delay from clock to output valid
44
45 set savePath        {../VERILOG/}
46 #####
47 ##      below here shouldn't need to be changed...      ##
48 #####
49
50 ## Analyze and Elaborate the HDL files
51 read_hdl    -vhdl    ${hdlFiles}
52 elaborate   ${basename}
53
54 ## Apply Constraints and generate clocks
55 set clock [define_clock -period ${clkPeriod_ps} -name ${clkName} [
56     clock_ports]]
57 external_delay -input $clkInDelay_ps -clock ${clkName} [find / -port
58     ports_in/*]
59 external_delay -output $clkOutDelay_ps -clock ${clkName} [find /
60     -port ports_out/*]
61
62 ## Sets transition to default values for Synopsys SDC format,
63 ## fall/rise 400ps
64 dc::set_clock_transition .4 $clkName
65
66 ## check that the design is OK so far
67 check_design -unresolved
68 report timing -lint
69
70 ## Synthesize the design to the target library
71 synthesize -to_mapped
72
73 ## Write out the reports
74 report timing > ${savePath}${basename}_${runName}_timing.rep
75 report gates > ${savePath}${basename}_${runName}_cell.rep
76 report power > ${savePath}${basename}_${runName}_power.rep
77
78 ## Write out the structural Verilog and sdc files
79 write_hdl -mapped > ${savePath}${basename}_${runName}.v
80 write_sdc > ${savePath}${basename}_${runName}.sdc

```

```

81  ## write sdf-timing file with
    ## -prec 3          : Use 3 digits floating point
    ##                  in delay calculation
83  ## -edges check_edge : Check which edge delays are defined in
    ##                  technology file and calculate only
85  ##                  those edges.
    write_sdf -prec 3 -edges check_edge > ${savePath}${basename}_${
        runName}.sdf
87
    ## log actual time in logfile
89  date

91  ## exit rc (usefull, for "rc -f <thisFileName>")
    quit

```

Listing 5.1: rc.tcl

5.2 counter.vhdl

```

1  library ieee;
    use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
    use ieee.std_logic_unsigned.all;
5
    entity counter is
6      port(
7          clk:    in std_logic;
9          nRst:   in std_logic;
            c_out:  out std_logic_vector(3 downto 0)
11         );
    end counter;
13
    architecture arch of counter is
15        signal c_int: std_logic_vector(3 downto 0);
    begin
17        process(clk)
            begin
19            if (clk'event and clk = '1') then
                if (nRst = '0') then
21                    c_int <= "0000"; -- others <= '0';
                else
23                    c_int <= c_int + "0001";
                end if;
25            end if;
        end process;
    end arch;

```



```

27     c_out <= c_int;
29 end arch;

```

Listing 5.2: VHDL-Code des Zählers

5.3 Testbench

```

library ieee;
2 use ieee.std_logic_1164.all;
  use ieee.std_logic_arith.all;
4 use ieee.std_logic_unsigned.all;

6 entity counter_tb is
end entity counter_tb;

8
architecture bhv of counter_tb is
10   component counter is
      port(
12       clk:    in std_logic;
          nRst: in std_logic;
14       c_out: out std_logic_vector(3 downto 0)
      );
16   end component;

18   signal clk: std_logic := '0';
      signal nRst: std_logic;

20   signal c_out: std_logic_vector(3 downto 0);

22
begin
24   dut: counter
      port map(
26       clk => clk ,
          nRst => nRst ,
28       c_out => c_out
      );

30
      clk <= not clk after 20 ns;
32       nRst <= '0', '1' after 100 ns;

34 end bhv;

```

Listing 5.3: Testbench des Zählers

5.4 counter.v (verilog-Code)

```
2 // Generated by Cadence Encounter(R) RTL Compiler RC14.25 - v14.20-
   s046_1
4 // Verification Directory fv/counter
6 module counter(clk , nRst , c_out);
   input clk , nRst;
8   output [3:0] c_out;
   wire clk , nRst;
10  wire [3:0] c_out;
   wire UNCONNECTED, UNCONNECTED0, UNCONNECTED1, UNCONNECTED2, n_0,
      n_1,
12      n_2, n_3;
   wire n_4, n_5, n_6, n_7, n_8, n_9, n_10, n_11;
14  DF3 \c_int_reg[3] (.C (clk), .D (n_11), .Q (c_out[3]), .QN
      (UNCONNECTED));
16  NOR21 g63 (.A (n_8), .B (n_10), .Q (n_11));
   DF3 \c_int_reg[2] (.C (clk), .D (n_9), .Q (c_out[2]), .QN
18      (UNCONNECTED0));
   XNR21 g65 (.A (c_out[3]), .B (n_5), .Q (n_10));
20  NOR21 g66 (.A (n_8), .B (n_7), .Q (n_9));
   INV3 g67 (.A (n_6), .Q (n_7));
22  DF3 \c_int_reg[1] (.C (clk), .D (n_4), .Q (c_out[1]), .QN
      (UNCONNECTED1));
24  ADD22 g68 (.A (n_1), .B (c_out[2]), .CO (n_5), .S (n_6));
   NOR21 g70 (.A (n_8), .B (n_3), .Q (n_4));
26  DF3 \c_int_reg[0] (.C (clk), .D (n_0), .Q (c_out[0]), .QN
      (UNCONNECTED2));
28  INV3 g71 (.A (n_2), .Q (n_3));
   ADD22 g72 (.A (c_out[1]), .B (c_out[0]), .CO (n_1), .S (n_2));
30  NOR21 g74 (.A (n_8), .B (c_out[0]), .Q (n_0));
   INV3 g75 (.A (nRst), .Q (n_8));
32 endmodule
```

Listing 5.4: Verilog-Code des Zählers

5.5 amsSetup.tcl - zusätzliche ams-Befehle für encounter

```
1 #####
   ##                                     #
```

```

3  ## Encounter Command File                                     #
4  ##                                                             #
5  ## Owner: austriamicrosystems                               #
6  ## HIT-Kit: Digital                                         #
7  ## version: 03-Nov-2009                                     #
8  ##                                                             #
9  #####
11 ## Global variables
12 set topcellname "none"
13 set dbdir "DB"
14
15 proc amsHelp {} {
16     print "#### Available Functions"
17     print "----#      - amsDbSetup..... Setup
Database - read Config"
18     print "----#      - amsUserGrid..... Sets the
grid for the IO-Cells"
19     print "----#      - amsGlobalConnect type..... connects
global nets: "
20     print "----#                                           type =
core | both"
21     print "----#      - amsAddEndCaps..... place Caps"
22     print "----#      - amsOpCond cond..... set
operating condintions: "
23     print "----#                                           cond =
typ | minmax | min | max"
24     print "----#      - amsFillcore ..... places core
filler cells "
25     print "----#      - amsFillperi ..... places
periphery filler cells "
26     print "----#      - amsRoute router..... run routing
with: "
27     print "----#                                           router
= nano | wroute | wroute2 (using 2CPUs)"
28     print "----#      - amsWrite postfix ..... writes GDS,
Verilog NL, SPEF, DB"
29     print "----#      - amsWriteSDF ..... write
combined SDF for all 3 cases"
30     print "----#      - amsWriteSDF4View viewList..... write SDF
for all analysis views in list"
31     print "----#      - amsZoomTo x y ..... zooms to
coordinates x y"
32     print "#### "
33 }

```

```

35 proc amsMakeChip {} {
    ##—— Load configuration file
37     amsDbSetup

39     ##—— Set User Grid
        amsUserGrid

41
43     ##—— make global connections
        amsGlobalConnect both

45 }

47 proc amsDbSetup {} {
    ##—— Load configuration file
49     loadConfig c35b3_std.conf 0
        commitConfig
51     setCTSMode -bottomPreferredLayer 2
        setMaxRouteLayer 3
53 }

55
57 proc amsUserGrid {} {
    ##—— Set user grids
        setPreference ConstraintUserXGrid 0.1
59     setPreference ConstraintUserXOffset 0.1
        setPreference ConstraintUserYGrid 0.1
61     setPreference ConstraintUserYOffset 0.1
        setPreference SnapAllCorners 1
63     setPreference BlockSnapRule 2

65     snapFPlanIO -usergrid
67 }

69 proc amsGlobalConnect type {
    ##—— Define global power connects
    switch $type {
71         "core" {
                ##—— Define global Power nets — make global
                connections
73                 clearGlobalNets
                    globalNetConnect vdd! -type pgpin -pin vdd! -inst *
                    -module {}
75                 globalNetConnect gnd! -type pgpin -pin gnd! -inst *
                    -module {}
                }
77         "both" {

```

```

79         ##--- Define global Power nets - make global
connections
    clearGlobalNets
    globalNetConnect vdd! -type pgpin -pin vdd! -inst *
81 -module {}
    globalNetConnect gnd! -type pgpin -pin gnd! -inst *
    -module {}
        #globalNetConnect vdd3o! -type pgpin -pin vdd3o!
    -inst * -module {}
83        #globalNetConnect vdd3r1! -type pgpin -pin vdd3r1!
    -inst * -module {}
        #globalNetConnect vdd3r2! -type pgpin -pin vdd3r2!
    -inst * -module {}
85        #globalNetConnect gnd3o! -type pgpin -pin gnd3o!
    -inst * -module {}
        #globalNetConnect gnd3r! -type pgpin -pin gnd3r!
    -inst * -module {}
87        }
    }
89 }

91 proc amsOpCond cond {

93     switch $cond {
        "typ" {
95         setOpCond -min TYPICAL -minLibrary c35_CORELIB \
            -max TYPICAL -maxLibrary c35_CORELIB
97         # -powerDomain normal
98         # setOpCond -min TYPICAL -minLibrary c35_CORELIB \
99         # -max TYPICAL -maxLibrary c35_CORELIB \
100        # -powerDomain modulator
101     }
        "minmax" {
103         setOpCond -min BEST-MIL -minLibrary c35_CORELIB \
            -max WORST-MIL -maxLibrary c35_CORELIB
105         # -powerDomain normal
106         # setOpCond -min BEST-MIL -minLibrary
107         c35_CORELIB_3B_1.8V_min \
            -max WORST-MIL -maxLibrary c35_CORELIB_3B_1.8V_max
108         # \
109         # -powerDomain modulator
110     }
        "min" {
111         setOpCond -min BEST-MIL -minLibrary c35_CORELIB \
            -max BEST-MIL -maxLibrary c35_CORELIB
113     }
    }
}

```

```

115         "max" {
116             setOpCond -min WORST-MIL -minLibrary c35_CORELIB \
117                 -max WORST-MIL -maxLibrary c35_CORELIB
118         }
119     }
120 }
121
122 proc amsAddEndCaps {} {
123     ##-- add CAP cells
124     addEndCap -preCap ENDCAPL -postCap ENDCAPR -prefix ENDCAP
125 }
126
127 proc amsFillcore {} {
128     ##-- Add Core Filler cells
129     source fillcore.tcl
130     ## or
131     ##addFiller -cell FILL25 FILL10 FILL5 FILL2 FILL1 -prefix FILLER
132     ##addFiller -cell FILLRT25 FILLRT10 FILLRT5 FILLRT2 FILLRT1
133     -prefix FILLERRT
134 }
135
136 proc amsFillperi {} {
137     ##-- Add Peri Filler cells
138     source fillperi.tcl
139 }
140
141 proc amsRoute {{router wroute}} {
142     switch $router {
143         "nano" {
144             ##-- Run Routing
145             ##-- Nano-Route
146             getNanoRouteMode -quiet
147             getNanoRouteMode -quiet envSuperThreading
148             setNanoRouteMode -quiet -drouteFixAntenna true
149             setNanoRouteMode -quiet -routeInsertAntennaDiode
150
151             false
152
153             setNanoRouteMode -quiet -timingEngine CTE
154             setNanoRouteMode -quiet -routeWithTimingDriven false
155             setNanoRouteMode -quiet -routeWithEco false
156             setNanoRouteMode -quiet -routeWithSiDriven false
157             setNanoRouteMode -quiet -routeTdrEffort 2
158             setNanoRouteMode -quiet -routeSiEffort normal
159             setNanoRouteMode -quiet -routeWithSiPostRouteFix
160
161             false
162
163             setNanoRouteMode -quiet -drouteAutoStop true
164         }
165     }
166 }

```

```

157         setNanoRouteMode -quiet -routeSelectedNetOnly false
158         setNanoRouteMode -quiet -drouteStartIteration default
159         setNanoRouteMode -quiet -envNumberProcessor 1
160         setNanoRouteMode -quiet -drouteEndIteration default
161         globalDetailRoute
162     }
163     "wroute" {
164         ###--- WROUTE
165         wroute
166     }
167     "wroute2" {
168         ###--- WROUTE
169         wroute -multiCpu 2
170     }
171 }
172
173 proc amsSave postfix {
174     global topcellname
175     global dbdir
176     set filename [format "%s/%s_%s.enc" $dbdir $topcellname $postfix]
177     saveDesign $filename
178 }
179
180 proc amsWrite postfix {
181     global topcellname
182     ###--- Save Design
183     amsSave $postfix
184     ###--- Write GDS2
185     set filename [format "%s_%s_fe.gds" $topcellname $postfix]
186     streamOut $filename -mapFile gds2.map -libName DesignLib
187     -structureName $topcellname \
188         -attachInstanceName -attachNetName -stripes 1 -units 1000
189     -mode ALL
190
191     ###--- Verilog Netlist
192     set filename [format "%s_%s.v" $topcellname $postfix]
193     saveNetlist $filename
194
195     ###--- Extract detail parasitics
196     set filename [format "SDF/%s_%s.rcdb" $topcellname $postfix]
197     # for SOC version < 7.1
198     # setExtractRCMode -detail -rcdb $filename -relative_c_t 0.01
199     -total_c_t 5.0 -reduce 5 -noise
200     setExtractRCMode -engine detail -rcdb $filename
201     setXCapThresholds -totalCThreshold 5.0 -relativeCThreshold 0.01

```

```

extractRC
201 set filename [format "SDF/%s_%s.spef" $topcellname $postfix]
    rcOut -spef $filename
203
205 ##-- run QX extraction
runQRC -lefFileList {} -layerMapping {} -grayData obs
207 set filename [format "SDF/%s_%s_qrc.spef" $topcellname $postfix]
    rcOut -spef $filename
209 }
211
213 proc amsWriteSDF {} {
    global topcellname
    ##-- Parasitic Extraction
215 #runQX
217 ##-- typical SDF
amsOpCond typ
219 set filename_t [format "SDF/%s_typ.sdf" $topcellname]
#delayCal -sdf $filename_t
221 write_sdf -version 2.1 -prec 3 -edges check_edge
    -average_typ_delays \
        -remashold -splitrecrem -splitsetuphold -force_calculation \
223 $filename_t
225 ##-- best case SDF
amsOpCond min
227 setAnalysisMode -checkType hold
set filename_b [format "SDF/%s_best.sdf" $topcellname]
229 #delayCal -sdf $filename_b
write_sdf -early -version 2.1 -prec 3 -edges check_edge
    -average_typ_delays \
231 -remashold -splitrecrem -splitsetuphold -force_calculation \
    $filename_b
233
235 ##-- worst case SDF
amsOpCond max
setAnalysisMode -checkType setup
237 set filename_w [format "SDF/%s_worst.sdf" $topcellname]
#delayCal -sdf $filename_w
239 write_sdf -late -version 2.1 -prec 3 -edges check_edge
    -average_typ_delays \
241 -remashold -splitrecrem -splitsetuphold -force_calculation \
    $filename_w

```



```

243  ###-- Combine all SDFs
      set filename [format "SDF/%s_all.sdf" $topcellname]
245  sdfCombine -file $filename_b $filename_t $filename_w -output
      $filename
      print "### Combined SDF File for best/typ/worst written!!"
247  }

249  ###-- write SDF for a specific analysis view
proc amsWriteSDF4View {viewList} {
251  global topcellname

253  foreach view $viewList {
      set filename [format "SDF/%s_%s.sdf" $topcellname $view]
255  print "----# Analysis View: $view\n"

257  write_sdf -version 2.1 -prec 3 -edges check_edge
      -average_typ_delays \
          -remashold -splitrecrem -splitsetuphold -force_calculation \
259  -view $view $filename
      print "----# Created SDF: $filename\n"
261  }
  }

263  ###-- Other usefule procedures
265
proc amsZoomTo {x y {factor 10}} {
267  set llx [expr {$x - $factor}]
      set lly [expr {$y - $factor}]
269  set urx [expr {$x + $factor}]
      set ury [expr {$y + $factor}]
271  zoomBox $llx $lly $urx $ury
  }

273  ###-- End of First Encounter TCL command file

275  proc protoSDF {} {
      amsDbSetup
277  floorplan -r 1.0 0.8 2 2 2 2
      setPlaceMode -fp true -timingDriven false -reorderScan false
      -doCongOpt false -modulePlan false
279  placeDesign -noPrePlaceOpt
      trialRoute -maxRouteLayer 3 -floorplanMode
281  extractRC
      amsWriteSDF
283  }

```

Listing 5.5: amsSetup.tcl für encounter

5.6 c35b3.conf - Design-Konfigurationsdatei

```
#####  
2 # #  
# FirstEncounter Input configuration file #  
4 # #  
# Owner: austriamicrosystems #  
6 # HIT-Kit: Digital #  
# version: 07-Aug-2012 #  
8 # #  
#####  
10 global rda_Input  
set cwd CWD  
12 set rda_Input(import_mode) {-treatUndefinedCellAsBbox 0  
-keepEmptyModule 1 }  
set rda_Input(ui_netlist) "VERILOG/counter_syn.v"  
14 set rda_Input(ui_netlisttype) {Verilog}  
set rda_Input(ui_ilmlist) {}  
16 set rda_Input(ui_settop) {1}  
set rda_Input(ui_topcell) {counter}  
18 set rda_Input(ui_celllib) {}  
set rda_Input(ui_iolib) {}  
20 set rda_Input(ui_areaiolib) {}  
set rda_Input(ui_blklib) {}  
22 set rda_Input(ui_kboxlib) {}  
set rda_Input(ui_gds_file) {}  
24 set rda_Input(ui_timelib,min) {}  
set rda_Input(ui_timelib,max) {}  
26 set rda_Input(ui_timelib) "/home/cdsmgr/ams/liberty/c35_3.3V/  
c35_CORELIB_BC.lib \\  
c35_CORELIB_WC.lib \\  
28 c35_CORELIB_TYP.lib \\  
c35_IOLIB_TYP.lib \\  
30 c35_IOLIB_BC.lib \\  
c35_IOLIB_WC.lib \\  
32 " "  
set rda_Input(ui_smodDef) {}  
34 set rda_Input(ui_smodData) {}  
set rda_Input(ui_dpath) {}  
36 set rda_Input(ui_tech_file) {}  
set rda_Input(ui_io_file) {}
```

```

38 set rda_Input(ui_timingcon_file) {}
set rda_Input(ui_latency_file) {}
40 set rda_Input(ui_scheduling_file) {}
set rda_Input(ui_buf_footprint) {}
42 set rda_Input(ui_delay_footprint) {}
set rda_Input(ui_inv_footprint) {}
44 set rda_Input(ui_leffile) {}
set rda_Input(ui_core_cntl) {aspect}
46 set rda_Input(ui_aspect_ratio) {1.0}
set rda_Input(ui_core_util) {0.650}
48 set rda_Input(ui_core_height) {}
set rda_Input(ui_core_width) {}
50 set rda_Input(ui_core_to_left) {50}
set rda_Input(ui_core_to_right) {50}
52 set rda_Input(ui_core_to_top) {50}
set rda_Input(ui_core_to_bottom) {50}
54 set rda_Input(ui_max_io_height) {0}
set rda_Input(ui_row_height) {}
56 set rda_Input(ui_isHorTrackHalfPitch) {0}
set rda_Input(ui_isVerTrackHalfPitch) {1}
58 set rda_Input(ui_ioOri) {R0}
set rda_Input(ui_isOrigCenter) {0}
60 set rda_Input(ui_exc_net) {}
set rda_Input(ui_delay_limit) {1000}
62 set rda_Input(ui_net_delay) {1000.0ps}
set rda_Input(ui_net_load) {0.5pf}
64 set rda_Input(ui_in_tran_delay) {0.1ps}
set rda_Input(ui_captbl_file) "-typical /home/cdsmgr/ams/cds/HK_C35/
LEF/encounter/c35b3-typical.capTable \
66 -best /home/cdsmgr/ams/cds/HK_C35/LEF
/encounter/c35b3-best.capTable \
-worst /home/cdsmgr/ams/cds/HK_C35/
LEF/encounter/c35b3-worst.capTable"
68 set rda_Input(ui_defcap_scale) {1.0}
set rda_Input(ui_detcap_scale) {1.0}
70 set rda_Input(ui_xcap_scale) {1.0}
set rda_Input(ui_res_scale) {1.0}
72 set rda_Input(ui_shr_scale) {1.0}
set rda_Input(ui_time_unit) {none}
74 set rda_Input(ui_cap_unit) {}
set rda_Input(ui_oa_reflib) "TECH_C35B3 \
76 CORELIB \
IOLIB_3M \
78 "
set rda_Input(ui_oa_abstractname) {abstract}
80 set rda_Input(ui_oa_layoutname) {layout}

```

```

set rda_Input(ui_sigstormlib) {}
82 set rda_Input(ui_cdb_file) {}
set rda_Input(ui_echo_file) {}
84 set rda_Input(ui_xilm_file) {}
set rda_Input(ui_qxtech_file) {/home/cdsmgr/ams/assura/c35b3/c35b3/
    RCX-typical/qrcTechFile}
86 set rda_Input(ui_qxlayermap_file) {/home/cdsmgr/ams/cds/HK_C35/LEF/
    c35b3/qrcly.map}
set rda_Input(ui_qxlib_file) {}
88 set rda_Input(ui_qxconf_file) {}
set rda_Input(ui_pwrnet) {vdd! \
90     vdd3r1! vdd3r2! vdd3o! \
    }
92 set rda_Input(ui_gndnet) {gnd! \
    gnd3r! gnd3o! \
94     }
set rda_Input(flip_first) {1}
96 set rda_Input(double_back) {1}
set rda_Input(assign_buffer) {0}
98 set rda_Input(ui_pg_connections) ""
set rda_Input(ui_gen_footprint) {1}

```

Listing 5.6: c35b3_*.conf für encounter

5.7 Skript für einfachen Login

```

1 #!/bin/bash

3 # This script mounts the cadence server via sshfs and also logs into
# the cadence server via ssh -X. The ssh-agent (ssh-add) is used to
5 # keep the id_rsa file unlocked so the password has to be entered
    only
# once per session (boot).
7 # To unmount the remote directory type
# fusermount -u /home/<username>/cadence_server

9
if id | grep -q "root"; then
11     echo "script must be run as normal user."
    echo "exiting"
13     exit 1
fi

15
username="$(whoami)"
17 local_username="$(whoami)"
realname="$(getent passwd $username | cut -d: -f5 | cut -d, -f1)"

```

```

19 id_rsa_dir="/home/$local_username/.ssh"
server="141.22.14.104"
21 local_dir="/home/$local_username/cadence_server"
remote_dir="/home"
23
24 if [ "$username" != "$realname" ]; then
25     firstname=$(echo $realname | cut -f1 -d" ")
26     lastname=$(echo $realname | cut -f2 -d" ")
27
28     # create the assumed username on cadence server
29     username=$(echo $firstname | head -c 1)
30     username=$username$lastname
31     username=$(echo $username | awk '{print tolower($0)}')
32     echo "Using '$username' as login for cadence server ($server)."

```

```

59 fi
61 # check if the choosen identity file is owned by the user
file_owner=$(stat -c '%U' $id_rsa_file)
63 if [ "$file_owner" != "$local_username" ]; then
    echo "You need to change the ownership of the identity file
    $id_rsa_file!"
65     echo "exiting"
    exit 3
67 fi
69 # permission check of the identity file
permissions=$(stat -c '%a' $id_rsa_file)
71 permissions_g_o=${permissions:1:2}
if [ "$permissions_g_o" != "00" ]; then
73     echo "The permissions of $id_rsa_file are '$permissions' which
    allows others to read it! SSH will reject this file!"
    read -n 1 -p "Do you want to change the permissions to '600' now
    ? (y/n)" answer
75     echo
    case ${answer:0:1} in
77         y|Y )
            chmod 600 $id_rsa_file
79             ;;
            * )
81                 echo "exiting"
                exit 4
83             ;;
        esac
85 fi
unset $answer
87 # check if mount point exists
89 if [ ! -d $local_dir ]; then
    read -n 1 -p "Mount point doesn't exist. Create directory '
    cadence_server' in '/home/$local_username? (y/n)" answer
91     echo
    case ${answer:0:1} in
93         y|Y )
            mkdir $local_dir
95             test -d $local_dir || (echo "an error occured." && exit
            5)
97             ;;
            * )
                echo "exiting"
99                 exit 6

```

```

101         ;;
102     esac
103 fi
104 # check if it is already mounted
105 if [ ! -d $local_dir/$username ]; then
106
107     # check if the ssh-agent has any identities stored
108     if ssh-add -l | grep -q "no identities" ; then
109         ssh-add $id_rsa_file
110     fi
111
112     # mount the remote directory
113     sshfs $username@$server:$remote_dir $local_dir -o IdentityFile=
114     $id_rsa_file
115 fi
116 # ssh into the cadence server
117 ssh -X $username@$server -i $id_rsa_file

```

Listing 5.7: run_cadence.sh

```

1 [Desktop Entry]
2 Name=run Cadence
3 Comment=ssh into Cadence server and mount remount filesystem
4 Exec=/usr/bin/run_cadence.sh
5 Icon=/home/tlattmann/.local/share/applications/cadence_icon.jpg
6 Terminal=true
7 Type=Application

```

Listing 5.8: run_cadence.desktop

CD

Selbstständigkeitserklärung

Hiermit versichere ich, Thomas Lattmann, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM 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, 23.4.2018,

Unterschrift