

# 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

Betreuender Prüfer: Prof. Dr.-Ing. Karl-Ragmar Riemschneider  
Zweitgutachter: Prof. Dr.-Ing. Jürgen Vollmer

Abgegeben am 20.04.2018

# Inhaltsverzeichnis

<b>1</b>	<b>Abkürzungsverzeichnis</b>	<b>III</b>
<b>2</b>	<b>Einleitung</b>	<b>1</b>
2.1	Motivation . . . . .	1
2.2	Stand der Technik . . . . .	1
2.3	Ziel dieser Arbeit . . . . .	1
<b>3</b>	<b>Grundlagen</b>	<b>2</b>
3.1	Binäre Zahlendarstellung von Festkommazahlen . . . . .	2
3.1.1	Integer im 1er-Komplement . . . . .	2
3.1.2	Integer im 2er-Komplement . . . . .	2
3.1.3	SQ-Format im 2er-Komplement . . . . .	3
3.2	Folgen der Bitbegrenzung . . . . .	3
3.2.1	Maximale Auflösung . . . . .	3
3.2.2	Rauschen . . . . .	3
3.3	Fourierreihenentwicklung . . . . .	3
3.4	Fouriertransformation . . . . .	5
3.5	Diskrete Fouriertransformation (DFT) . . . . .	5
3.5.1	Verwendung . . . . .	5
3.5.2	Summen- und Matrizenschreibweise der DFT . . . . .	5
3.5.3	Inverse DFT . . . . .	6
3.6	Diskrete Kosinus Transformation (DCT) . . . . .	6
<b>4</b>	<b>Analyse</b>	<b>7</b>
4.1	Überlegungen DCT vs. DFT . . . . .	7
4.2	Wahl einer geeigneten DFT-Größe . . . . .	7
4.2.1	Bewertung verschiedener Twiddlefaktor-Matrizen . . . . .	7
4.3	Abschätzung des Rechenaufwandes . . . . .	10
4.3.1	Komplexe Multiplikation . . . . .	10
4.3.2	Matrizenmultiplikation . . . . .	10
4.3.3	Gegenüberstellung Butterfly / Matrixmultiplikation . . . . .	11
4.3.4	Gegenüberstellung reelle / komplexe Eingangswerte . . . . .	11
4.3.5	Anzahl der benötigten Multiplikationen . . . . .	11
4.3.6	Optimierte Matrixmultiplikation bezogen auf 8x8 . . . . .	11
<b>5</b>	<b>Entwurf</b>	<b>12</b>
5.1	Test der Matrizenmultiplikation . . . . .	12
5.2	Implementierung des Konstantenmultiplizierers . . . . .	12
5.2.1	Syntheseresultat eines 12 Bit Konstantenmultiplizierers . . . . .	12
5.3	Entwickeln der 2D-DFT . . . . .	13
5.4	Direkte Weiterverarbeitung der Zwischenergebnisse . . . . .	13
5.4.1	Matrizenmultiplikation . . . . .	13

5.4.2 Veranschaulichung 2D-DFT als Matrizenmultiplikation . . . . .	14
5.5 Optimieren der 8x8-DFT . . . . .	14
5.6 Ungleiche Bitbreiten bei gerader / ungerader Zeile . . . . .	15
5.7 Struktogramm . . . . .	15
5.8 Automatengraf . . . . .	17
<b>6 Evaluation</b>	<b>21</b>
6.1 Simulation . . . . .	21
6.1.1 NC Sim - positive Zahlendarstellung . . . . .	21
6.1.2 Anzahl benötigter Takte . . . . .	21
6.1.3 Akkumulation von Fehlern . . . . .	21
6.2 Testumgebung . . . . .	22
6.2.1 1D-DFT, Integer . . . . .	22
6.2.2 2D-DFT, Integer . . . . .	22
6.2.3 2D-DFT, Q-Format . . . . .	22
6.2.4 Struktogramm des Testablaufs . . . . .	22
6.2.5 Reale Eingangswerte . . . . .	22
6.3 Anzahl Standardzellen . . . . .	22
6.3.1 1D / 2D . . . . .	22
6.3.2 3 Lagen / 4 Lagen . . . . .	22
6.4 Visualisierung der Netzliste . . . . .	22
6.5 Floorplan, Pading . . . . .	22
<b>7 Schlussfolgerungen</b>	<b>24</b>
7.1 Zusammenfassung . . . . .	24
7.2 Bewertung und Fazit . . . . .	24
7.3 Ausblick . . . . .	24
<b>Abbildungsverzeichnis</b>	<b>25</b>
<b>Tabellenverzeichnis</b>	<b>26</b>
<b>Literatur</b>	<b>27</b>
<b>8 Anhang</b>	<b>28</b>
8.1 Skript zur Bewertung von Twiddlefaktormatrizen . . . . .	28
8.2 Gate-Report des 12 Bit Konstantenmultiplizierers . . . . .	29
8.3 Twiddlefaktormatrix im S1Q10-Format . . . . .	30
8.4 Ausmultiplizierte Matrizen . . . . .	34
8.5 Ausmultiplizieren der 8x8 DFT . . . . .	34
8.6 Programmcode . . . . .	44
8.7 Testumgebung . . . . .	64

# 1 Abkürzungsverzeichnis

<b>1D-DFT</b>	Eindimensionale Diskrete Fouriertransformation
<b>2D-DFT</b>	Zweidimensionale Diskrete Fouriertransformation
<b>ADC</b>	Analog Digital Converter
<b>ADU</b>	Analog Digital Umsetzer
<b>AMR</b>	anisotroper magnetoresistiver Effekt
<b>ASIC</b>	Application Specific Integrated Circuit, <i>dt.: Anwendungsspezifischer Integrierter Schaltkreis</i>
<b>DFT</b>	Diskrete Fouriertransformation
<b>FFT</b>	Fast Fouriertransformation
<b>FT</b>	Fouriertransformation
<b>IDFT</b>	Inverse Diskrete Fouriertransformation
<b>ISAR</b>	Integrated Sensor Array
<b>LSB</b>	Least Significant Bit
<b>MSB</b>	Most Significant Bit
<b>TMR</b>	tunnelmagnetoresistiver Effekt

## 2 Einleitung

### 2.1 Motivation

### 2.2 Stand der Technik

Der verwendete Prozess ist mit  $350\text{ }\mu\text{m}$  im Vergleich zu modernen Prozessen mit beispielsweise  $20\text{ nm}$  Strukturbreite um die Größenordnung  $10^4$  größer. Entsprechend handelt es sich um einen relativ alten Prozess.

Kurze Beschreibung zu Standardzellen.

### 2.3 Ziel dieser Arbeit

Im Rahmen des Integrated Sensor Array (ISAR)-Projekts der HAW Hamburg soll zur Signalvorverarbeitung einer Matrix von Magnetsensoren eine Zweidimensionale Diskrete Fouriertransformation (2D-DFT) in VHDL implementiert werden. Mit der 2D-DFT sollen relevante Signalanteile identifiziert werden, um so den Informationsgehalt der Sensorsignale zu reduzieren. Die Sensoren basieren auf dem anisotropen magnetoresistiven Effekt (AMR)- bzw. in einem späteren Schritt tunnelmagnetoresistiven Effekt (TMR).

In einem Text zitiert dann so [1, S. 10-20] und blabla.

## 3 Grundlagen

### 3.1 Binäre Zahlendarstellung von Festkommazahlen

#### 3.1.1 Integer im 1er-Komplement

Bei der Interpretation des Bitvektors als Integer im Einerkomplement werden die Bits anhand ihrer Position im Bitvektor gewichtet, wobei das niederwertigste Bit (LSB, least significant bit) dem Wert für den Faktor  $2^0$  entspricht, das Bit links davon 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 es sich um eine negative oder positive Zahl handelt. Dies hat zur Folge, dass es eine positive und eine negative Null und somit eine Doppeldeutigkeit gibt. Desweiteren wird ein LSB an Auflösung verschenkt. Der Wertebereich erstreckt sich von  $-2^{MSB-1} + 1 \text{ LSB}$  bis  $2^{MSB-1} - 1 \text{ LSB}$ .

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 Multiplikanden ermitteln. Die eigentliche Multiplikation beschränkt sich auf die Bits MSB-1 bis LSB. Da als einziger konstanter Multiplikand in der 8x8-DFT-Matrix der Faktor  $\pm \frac{\sqrt{2}}{2}$  auftaucht, also das oben angeführte Beispiel zutrifft, erschien diese Darstellungsform zwischenzeitlich interessant.

Nachteile zeigen sich hingegen bei der Addition sowie Subtraktion negativer Zahlen. Auch hierfür gibt es schematische Rechenregeln, diese erfordern jedoch mehr Zwischenschritte als im Zweierkomplement. Darüberhinaus ist dieses Verfahren aufgrund der geringen Bedeutung in keiner VHDL-Bibliothek implementiert. (Verifizieren!)

#### 3.1.2 Integer im 2er-Komplement

Bei der Interpretation als Zweierkomplement kann anhand des MSB ebenfalls erkannt werden, ob es sich um eine positive oder negative Zahl handelt. Dennoch wird es nicht als Vorzeichenbit gewertet. Viel mehr bedeutet ein gesetztes MSB  $-2^{MSB-1}$ , welches der negativsten darstellbaren Zahl entspricht. Hierbei sind alle anderen Bits auf 0. 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  $2^{MSB-1} - 1 \text{ LSB}$  darstellen. Der positive Wertebereich ist also um ein LSB kleiner als der negative und es gibt keine doppelte Null.

Um das Vorzeichen umzukehren müssen alle Bits invertiert werden. Auf den neuen Wert muss abschließend 1 LSB addiert werden.

Vorteile bei dieser Darstellung ist, dass die mathematischen Operationen Addition, Subtraktion und Multiplikation direkt angewandt werden können. Unterstützt werden

sie z.B. von den Datentypen `unsigned` sowie `signed`, welche in der Bibliothek u.a. `ieee.numeric_std.all` definiert sind.

### 3.1.3 SQ-Format im 2er-Komplement

Im SQ-Format werden Zahlen als vorzeichenbehafteter Quotient (signed quotient) dargestellt. Die konkretere Schreibweise von beispielsweise S1Q10 bedeutet, dass zusätzlich zu einem Vorzeichenbit noch ein weiteres Bit vor dem Komma steht. Für den Quotient stehen 10 Bit zur Verfügung, was einer maximalen Auflösung von  $1\text{ LSB} = 2^{-10} = \frac{1}{1024} = 9,765625 \cdot 10^{-4}$  entspricht. Der Wertebereich liegt in diesem Fall liegt bei  $-2$  bis  $1,999023438$ . Er wurde in der vorliegenden Arbeit so gewählt, da sich hiermit die Werte  $\pm 3,3\text{ V}/2 = \pm 1,65\text{ V}$  darstellen lassen, was nach Abzug des Offsets den Eingangsspannungen des Analog Digital Converter (ADC) von  $0\text{ V}$  bis  $3,3\text{ V}$  entspricht und zum derzeitigen Stand des Projekts davon ausgegangen wird, dass der verwendete ADC Werte mit zwölf Bit Breite ausgibt. Es wird von einer Vorverarbeitung ausgegangen, die dies erledigt.

## 3.2 Folgen der Bitbegrenzung

### 3.2.1 Maximale Auflösung

Um einen guten Kompromiss aus ausreichender Genauigkeit, Geschwindigkeit und Platzbedarf zu erzielen, wird von Eingangs- / Ausgangssignalen mit 12 Bit Breite zwischen den einzelnen Komponenten auf dem Chip ausgegangen.

Sicherlich ist eine hohe Genauigkeit erstrebenswert. Es gilt jedoch zu bedenken, dass mit höheren Bitbreiten auch der Platzbedarf jedes einzelnen Datensignals aufgrund der zusätzlich benötigten Leitungen sowie der Flip-Flops für die (Zwischen-) Speicherung, linear steigt. Bei Additionen und insbesondere Multiplikationen geht mit jedem zusätzlichen Bit ebenfalls ein linear steigender Zeitbedarf einher. Eine Bitbreite von größer 24 Bit (bei Eingangsspannungen kleiner  $5\text{ V}$ ) ist darüber hinaus bei ADC nicht sinnvoll, da durch thermisches Rauschen die ermittelten Werte beeinflusst werden und die Pegel des Rauschen in dieser Größenordnung liegen. Derzeit wird davon ausgegangen, dass der Chip in einer Strukturgröße von  $350\text{ nm}$  gefertigt wird, sodass sich jeder zusätzliche Platzbedarf merklich auswirkt.

Akkumulation von Fehlern kurz ansprechen.

### 3.2.2 Rauschen

## 3.3 Fourierreihenentwicklung

Mit einer Fourierreihe kann ein periodisches, abschnittsweise stetiges Signal aus einer Summe von Sinus- und Kosinusfunktionen zusammengesetzt werden. Die Schreibweise als Summe von Sinus- und Kosinusfunktionen (Gl. 3.1) ist eine der häufigsten Darstellungsformen.

$$x(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos(kt) + b_k \sin(kt)) \quad (3.1)$$

Die Fourierkoeffizienten lassen sich über die Gleichungen (3.2) und (3.3) berechnen:

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} x(t) \cdot \cos(kt) dt \quad \text{für } k \geq 0 \quad (3.2)$$

$$b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} x(t) \cdot \sin(kt) dt \quad \text{für } k \geq 1 \quad (3.3)$$

Mit der Exponentialschreibweise lassen sich Sinus und Kosinus auch wie in (3.4) und (3.5) ausdrücken:

$$\cos(kt) = \frac{1}{2} (e^{jkt} + e^{-jkt}) \quad (3.4)$$

$$\sin(kt) = \frac{1}{2j} (e^{jkt} - e^{-jkt}) \quad (3.5)$$

und zusammengefasst ergibt sich in (Gl. 3.6) der komplexe Zeiger, der eine Rotation im Gegenuhrzeigersinn auf dem Einheitskreis beschreibt. In Abbildung 4.1 dies zusätzlich noch grafisch dargestellt.

$$\begin{aligned} \cos(kt) + j \cdot \sin(kt) &= \frac{1}{2} (e^{jkt} + e^{-jkt}) + j \cdot \frac{1}{2j} (e^{jkt} - e^{-jkt}) \\ &= \frac{1}{2} (e^{jkt} + e^{jkt}) \\ &= e^{jkt} \end{aligned} \quad (3.6)$$

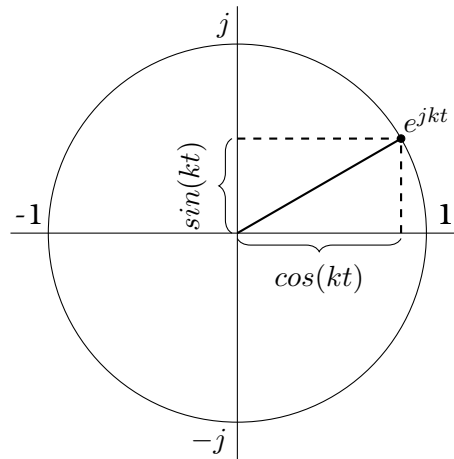


Abbildung 3.1: Einheitskreis, Zusammensetzung des komplexen Zeigers aus Sinus und Kosinus

Analog werden die Fourierkoeffizienten  $a_k$  und  $b_k$  als reellen bzw. imaginären Teil einer komplexen Zahl  $c_k$  betrachtet und lassen sich

$$x(t) = \sum_{-\infty}^{\infty} c_k e^{jkt} \quad (3.7)$$



### 3.4 Fouriertransformation

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

Grundlage für die Fouriertransformation ist das Fourierintegral (Gl. 3.8)

$$X(f) = \int_{-\infty}^{\infty} x(t) \cdot e^{-j2\pi ft} \quad (3.8)$$

Wenn Sinus und Kosinus wie in 3.4 und 3.5 als Exponentialfunktion geschrieben werden, können sie auch zu einer komplexen Exponentialfunktion zusammengefasst werden.

Für komplexere Signale, etwa ein Rechteck, ergeben sich entsprechend sehr viele dieser Peaks. Deren Höhe ist Information darüber, wie groß ihr Anteil, also die Amplitude des Zeitsignals, ist. Die Fouriertransformation kann als das Gegenteil der Fourierreihenentwicklung gesehen werden.

- unendliche Dauer? -> Leistungssignal?

Fourier-Transform: Zerlegung - endliche Dauer, Energiesignal

Energiesignal: Leistungssignal: Signal unendlicher Energie, aber mit endlicher mittlerer Leistung

Ein Zeitsignal hat ein eindeutig zuordbares Frequenzsignal (bijektiv), abgesehen von Amplitude? und Phase

Spektrum: Frequenzbestandteile eines Signals Berechnung des Spektrums: Spektralanalyse, Frequenzanalyse

Fourier-Synthese: Umkehrfunktion

In der Praxis, also basierend auf echten Messdaten, wird die Bestimmung des Spektrums Spektrumschätzung genannt.

### 3.5 Diskrete Fouriertransformation (DFT)

#### 3.5.1 Verwendung

Die Diskrete Fouriertransformation (DFT) (Gl. 3.9) ist die zeit- und wertdiskrete Variante der Fouriertransformation (FT), die statt von  $-\infty$  bis  $\infty$  nur von 0 bis  $N-1$  Elemente läuft. Im Frequenzspektrum wiederholt sich Da es sich um diskrete Werte handelt, geht das Integral in eine endliche Summe über. Die 2D-DFT ist nur möglich (sinnvoll), wenn die Eingangswerte in Form eines Vektors vorliegen.

#### 3.5.2 Summen- und Matrizenschreibweise der DFT

##### 1D-DFT

Die Eindimensionale Diskrete Fouriertransformation (1D-DFT) findet wie bereits erwähnt üblicherweise Anwendung, um vom Zeit- in den Frequenzbereich zu gelangen.

$$X^*[m] = \frac{1}{N} \cdot \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi mn}{N}} \quad (3.9)$$

Gleichung 3.11 zeigt die obige Summenformel umgeschrieben zu einer Matrixmultiplikation.

Mit Gleichung 3.10 werden zunächst alle Twiddlefaktoren in Matrixform berechnet, wobei  $n$  der Index des zu Berechnenden Elements des Vektors im Zeitbereich und  $m$  das Äquivalent im Frequenzbereich ist.

$$\sum_{m=0}^{N-1} \sum_{n=0}^{N-1} e^{-\frac{j2\pi mn}{N}} = W \quad (3.10)$$

Somit gilt:

$$X^* = W \cdot x \quad (3.11)$$

## 2D-DFT

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

$$\begin{aligned} X[u, v] &= \frac{1}{N} \sum_{n=0}^{N-1} X^*[m] \cdot e^{-\frac{j2\pi mn}{N}} \\ &= \frac{1}{MN} \sum_{m=0}^{M-1} \left( \sum_{n=0}^{N-1} f(m, n) \cdot e^{-\frac{j2\pi mn}{N}} \right) \cdot e^{-\frac{j2\pi mn}{M}} \end{aligned} \quad (3.12)$$

Auch hier lässt sich die Berechnung in Matrizenschreibweise darstellen:

$$\begin{aligned} X &= W \cdot x(t) \cdot W \\ &= X^* \cdot W \end{aligned} \quad (3.13)$$

### 3.5.3 Inverse DFT

Die Inverse Diskrete Fouriertransformation (IDFT) wird analog zur DFT mit

$$x[n] = \frac{1}{N} \sum_{m=0}^{N-1} X[m] \cdot e^{\frac{j2\pi mn}{N}} \quad (3.14)$$

beschrieben. Durch die umgekehrte Drehrichtung des komplexen Zeigers werden in der Matrizenschreibweise die Zeilen 1 und 7, 2 und 6 sowie 3 und 5 vertauscht.

## 3.6 Diskrete Kosinus Transformation (DCT)

## 4 Analyse

### 4.1 Überlegungen DCT vs. DFT

Die 2D-DFT findet häufig in der Bildverarbeitung Anwendung und lässt sich gut auf unser Problem übertragen.

### 4.2 Wahl einer geeigneten DFT-Größe

#### 4.2.1 Bewertung verschiedener Twiddlefaktor-Matrizen

In diesem Abschnitt werden verschiedene Größen von Twiddlefaktor-Matrizen auf ihre Werte untersucht und bewertet. Ziel ist es aus den in Frage kommenden jene zu ermitteln, die die trivialsten Berechnungen bei einer Multiplikation erfordert. Von Interesse sind aufgrund des dualen Zahlensystems Matrizen mit Werten, die sich einerseits mit wenigen Bits darstellen lassen und andererseits nur Bit-shifting zur Folge haben. Beide Anforderungen bedingen sich in der Regel gegenseitig.

In der folgenden Tabelle 4.1 werden die  $8 \times 8$ ,  $9 \times 9$ ,  $12 \times 12$ ,  $15 \times 15$  sowie  $16 \times 16$ -Matrix einander gegenüber gestellt. Da die Sensormatrix aus  $8 \times 8$  Sensoren aufgebaut ist, besteht ein Interesse an einer ungeraden Matrix. Dies hätte den Vorteil, dass sich über dem Mittelpunkt der Sensormatrix kein Element der Twiddlefaktormatrix befindet. Auf diese Weise ließe sich die ... einfacher ermitteln. Bekannt ist jedoch auch, dass die Fast Fouriertransformation (FFT) auf Matrizen mit den Abmessungen  $2^n$  basiert und es sich hierbei um ein sehr schnelles und effizientes Verfahren handelt. Deshalb werden auch Matrizen mit gerader Anzahl an Elementen untersucht. Die Beurteilung basiert auf dem Octave-Skript 8.1

Tabelle 4.1: Bewertung der 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
Verhältnis	3*	0,6875	3,5	0,3889	1

Als triviale Werte werden 0,  $\pm 0,5$  sowie  $\pm 1$  aufgefasst. Andere Werte die sich gut binär darstellen lassen tauchen nicht auf. Alle übrigen Werte werden als nicht trivial betrachtet, da eine Multiplikation mit ihnen eine komplexere Berechnung bedeutet.

Bei der  $8 \times 8$  Matrix gibt es, wie in Grafik 4.1 zu sehen, als nicht trivialen Wert mit  $|\sqrt{2}/2|$  für Real- und Imaginärteil nur einen einzigen Wert, welche dazu noch gemeinsam auftreten. Dies liegt daran, dass der Einheitskreis geachtet wird und für beispielsweise  $\frac{2 \cdot \pi}{8} = \frac{\pi}{4}$  Sinus und Cosinus identisch sind. Darüberhinaus ist dies auch der einzige Wert, der sowohl einen Real- als auch einen Imaginärteil besitzt. Alle anderen Faktoren haben in einem von beiden Teilen  $|1|$  und somit im anderen Teil 0.

In der bereits erwähnten Grafik 4.1 sind zur Veranschaulichung alle möglichen Zeiger der Twiddlefaktoren ( $W_{m,n}$ ) für die  $8 \times 8$  Matrix dargestellt. Berechnet werden diese mit der Gleichung 3.10, wobei es sich bei  $N$  um die Anzahl der Elemente im Vektor bzw. der Spalte einer Matrix von Werten im Zeitbereich handelt.  $n$  ist der Laufindex über die einzelnen Elemente,  $m$  das Äquivalent für den zu berechnenden Vektor (Matrixspalte) im Frequenzbereich. Beide fangen bei 0 an und laufen entsprechend bis  $N - 1$ .

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 können. Die andere Hälfte muss über die Bildung des 2er-Komplements lediglich negiert werden, was ein bedeutend geringerer Aufwand ist, als eine Multiplikation. Deshalb ist das berechnete Verhältnis von 3 in Tabelle 4.1 in Wirklichkeit deutlich höher und übertrifft mit 7 die  $12 \times 12$  Matrix um den Faktor 2. Dies gilt unter der Annahme, dass die Bildung des 2er-Komplements nicht berücksichtigt wird, was zumindest einer besseren Näherung entspricht, als es als eine volle Multiplikation zu werten.

Hierzu Abschnitt Abschätzung des Rechenaufwandes?

Anfangs wurde angenommen, dass das 1er-Komplement eine gute Wahl sein könnte, da hierbei die Darstellung negativer Zahlen einzig durch Setzen des vordersten Bit (Most Significant Bit (MSB)) erfolgt. Auf diese Weise könnte immer das selbe Resultat für den Imaginär- wie für den Realteil verwendet werden, das Vorzeichen würde sich über eine einfache XOR-Verknüpfung beider MSB ergeben. Diesem Vorteil steht jedoch eine komplexere Subtraktion (bzw. Addition negativer Zahlen) gegenüber. Der zusätzliche Aufwand entspricht relativ genau dem der Bildung des 2er-Komplements. Aus diesem Grund wurde sich für dieses entschieden, da es deutlich gängiger ist und weitere Vorteile bringt wie beispielsweise keine Doppeldeutigkeit durch eine negative Null hat.

In Abbildung 4.2 sind zur weiteren Veranschaulichung die komplexen Zeiger der Twiddlefaktoren dargestellt. Sie sind aufgeteilt auf 8 Einheitskreise, wobei jeder einen Laufindex ( $m$ ) des Zeitbereichs abdeckt. In den einzelnen Kreisen sind wiederum alle Laufindexe ( $n$ ) des Frequenzbereichs zu sehen.

Sollte ich diesen EK auch noch in Tikz machen?

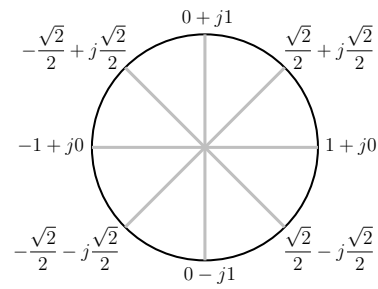


Abbildung 4.1: Einheitskreis mit relevanten Werten

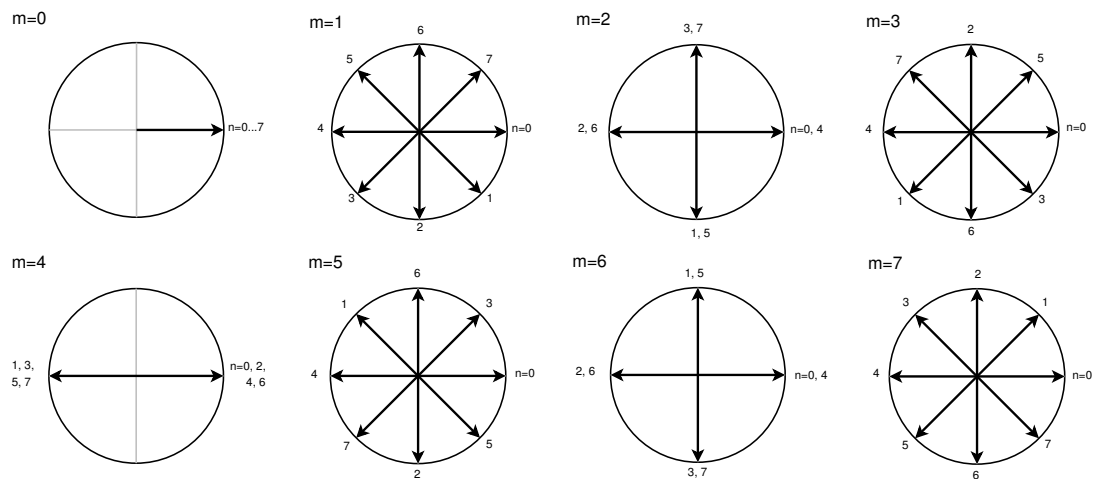
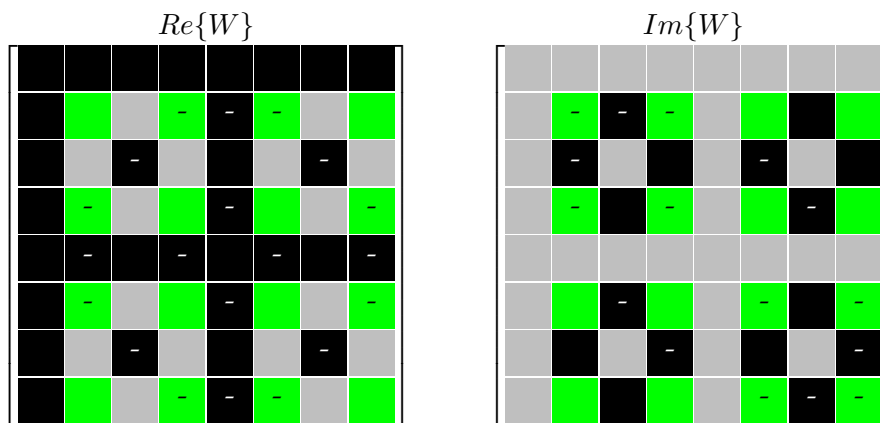
Abbildung 4.2: Twiddlefaktoren der  $8 \times 8$ -Matrix, aufgeteilt auf die Laufindexe

Abbildung 4.3: Matrizen-Darstellung der Twiddlefaktoren aufgeteilt nach Real- und Imaginärteil

Legende:  $\blacksquare = 1$   $\blacksquare = -1$   $\square = 0$   $\color{red}\square = \sqrt{2}/2$   $\color{red}\square = -\sqrt{2}/2$

Sowohl der Abbildung 4.2 als auch insbesondere der Darstellung 4.3 lassen sich

sehr gut die Symmetrien erkennen, die diese Twiddlefaktormatrix so vorteilhaft machen.

## 4.3 Abschätzung des Rechenaufwandes

### 4.3.1 Komplexe Multiplikation

Im allgemeinen Fall müssen gemäß Gl. 4.1 bei der komplexen Multiplikation vier einfache Multiplikation sowie zwei Additionen durchgeführt werden.

$$\begin{aligned} e + jf &= (a + jb) \cdot (c + jd) \\ &= a \cdot c + j(a \cdot d) + j(b \cdot c) + j^2(b \cdot d) \\ &= a \cdot c + b \cdot d + j(a \cdot d + b \cdot c) \end{aligned} \quad (4.1)$$

Da das Signal  $x_{sens}(t)$  der Sensoren rein reell ist, reduziert sich der Aufwand wie in Gl. 4.2 zu sehen auf zwei Multiplikationen und eine Addition.

$$\begin{aligned} e + jf &= a \cdot (c + jd) \\ &= a \cdot c + j(a \cdot d) \end{aligned} \quad (4.2)$$

Wie bereits unter 4.2.1 auf Seite 7 erörtert, kann sogar die komplexe Twiddlefaktormatrix in diesem speziellen Fall als rein reell betrachtet werden. Somit bleibt von der anfangs komplexen Multiplikation nur eine rein reelle Multiplikation und in 50% der Fälle die Bildung des 2er-Komplements übrig, was erheblich Rechenaufwand erspart.

$$X_{Sens}(f) = W \cdot x_{Sens}(t) \quad : \text{“rein reell”} \quad (4.3)$$

### 4.3.2 Matrizenmultiplikation

Die DFT kann als Summation oder als Matrixmultiplikation geschrieben werden, wobei sich letzteres auf einem Application Specific Integrated Circuit, *dt.: Anwendungsspezifischer Integrierter Schaltkreis* (ASIC) bedeutend besser implementieren lässt. Gleichung 4.4 stellt die DFT als Matrizenmultiplikation dar, wobei  $x(t)$  der Eingangsvektor (bzw. -matrix) im Zeitbereich und  $W$  die Twiddlefaktormatrix ist.

$$X(f) = F(k) = W \cdot x(t) = W \cdot f(m) \quad (4.4)$$

Betrachtet wird zunächst die Multiplikation eines Vektors mit einer Matrix.

$$X(f) = \mathcal{F}\{x(t)\} = \begin{bmatrix} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{bmatrix} = \begin{bmatrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{bmatrix} \cdot \begin{bmatrix} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{bmatrix}$$

$$X(f) = \mathcal{F}\{x(t)\} = \begin{bmatrix} \text{Matrix} \end{bmatrix} = \begin{bmatrix} \text{Matrix } W \end{bmatrix} \cdot \begin{bmatrix} \text{Matrix } x(t) \end{bmatrix}$$

$$X(f)' = F(k, l) = W \cdot F(k) = W \cdot x(t) \cdot W$$

$$X(f)' = \mathcal{F}\{X(f)\}' = \begin{bmatrix} \text{Matrix} \end{bmatrix} = \begin{bmatrix} \text{Matrix } X(f) \end{bmatrix} \cdot \begin{bmatrix} \text{Matrix } W \end{bmatrix}$$

#### 4.3.3 Gegenüberstellung Butterfly / Matrixmultiplikation

Die DFT wurde als Matrixmultiplikation implementiert, nachfolgend soll dies begründet und ein Vergleich beider Varianten erfolgen.

Zu einem frühen Zeitpunkt der Überlegungen an dieser Arbeit gab es noch die Idee die DFT so flexibel wie möglich zu halten, um unkompliziert auf andere Größen wechseln zu können. Hierfür sollten alle Koeffizienten der Twiddlefaktormatrix ladbar sein sowie die Größe der Matrix über eine globale Deklaration variabel gehalten werden. Diese Herangehensweise bedingt die Implementation als Matrixmultiplikation. Die Hoffnung der Projektgruppe bestand darin, dass das Synthesewerkzeug den VHDL-Code soweit optimiert, dass dies nicht händisch erfolgen müsste. Als klar war, dass die Optimierung nicht so tief greift, wurden die entsprechenden Schritte manuell umgesetzt.

Die Implementierung des Butterfly-Algorithmus nach Cooley und Tukey stellt eine effiziente Berechnung der DFT dar.

#### 4.3.4 Gegenüberstellung reelle / komplexe Eingangswerte

Die Sensormatrix liefert für jeden Pixel einen Sinus- und Kosinuswert. Diese können für die Berechnung der DFT zu einer komplexen Zahl zusammengefasst werden. Auf diese Weise lässt sich die Berechnung mathematisch kompakter schreiben. Dadurch, dass eine komplexe Multiplikation auf vier reellen Multiplikationen basiert, ist es jedoch möglich, dass die Anzahl reeller Multiplikationen hierdurch derer bei der getrennten Berechnung und anschließenden Zusammenführung übersteigt.

Bei der Berechnung der DFT

#### 4.3.5 Anzahl der benötigten Multiplikationen

#### 4.3.6 Optimierte Matrixmultiplikation bezogen auf 8x8

## 5 Entwurf

### 5.1 Test der Matrizenmultiplikation

Zunächst wurde die Berechnung als Ganzzahl-Multiplikation mit dem Faktor 3 betrachtet. Da es bei diesem Faktor und den gewählten Eingangswerten nicht zu einem Überlauf kommen kann, war es zu diesem Zeitpunkt noch nicht nötig, sich Gedanken über die Breite des Ergebnisvektors bzw den Ausschnitt daraus für die weitere Berechnung zu machen. Auch konnte an dieser Stelle noch auf den Bitshift zur Halbierung der Werte verzichtet werden.

Erst als der Faktor  $\frac{\sqrt{2}}{2}$  übernommen wurde, wurden die Ergebnisse breiter als der Vektor für die weitere Berechnung an Bits zur Verfügung stellt. Daraus folgt, dass ein Teil der Bits abgeschnitten werden müssen. Da die Dualzahlen jetzt im S1Q10-Format betrachtet werden, es sich also um Kommazahlen handelt, müssen die hinteren Bits abgeschnitten werden. Zudem können vorne Bits ohne Informationsverlust gestrichen werden, da durch die Multiplikation ein weiteres Negations-Bit dazugekommen ist und auf Grund des gegebenen Faktors der Wertebereich vorne nie ganz ausgenutzt wird. (Verifizieren / Belegen!)

### 5.2 Implementierung des Konstantenmultiplizierers

Anfangs wurde angenommen, dass Multiplikationen mit den Twiddlefaktoren  $\pm 1$  und  $\pm \frac{\sqrt{2}}{2}$  durchgeführt werden müssen. Dass bei einer optimierten 8x8-DFT wegen des expliziten ausprogrammierens der Berechnungen die Multiplikation mit  $\pm 1$  wegfällt, wurde recht schnell klar. Erst bei genauer Betrachtung der Twiddlefaktor-Matrix viel auf, dass in jeder Zeile gleich viele Additionen wie Subtraktionen vorhanden sind. Durch Umsortieren ist es dadurch möglich auf das Invertieren der Eingangswerte sowie den hierfür benötigten Takt und die Inverter zu verzichten. Weiter wird auch nur die Multiplikation mit  $+\frac{\sqrt{2}}{2}$  benötigt.

#### 5.2.1 Syntheseresultat eines 12 Bit Konstantenmultiplizierers

Tabelle 5.1: Vergleich Konstanten- mit regulärem Multiplizierer

	Konstantenmultiplizierer	regulärer Multiplizierer
Gatter	43	?
Inverter	10	?
Summe	53	?
Fläche (Prozess: 350nm)	4787 nm <sup>2</sup> ?	?

Der Ausgang hat so wie der Eingang 12 Bit, die niederwertigsten 12 Bit werden somit verworfen.



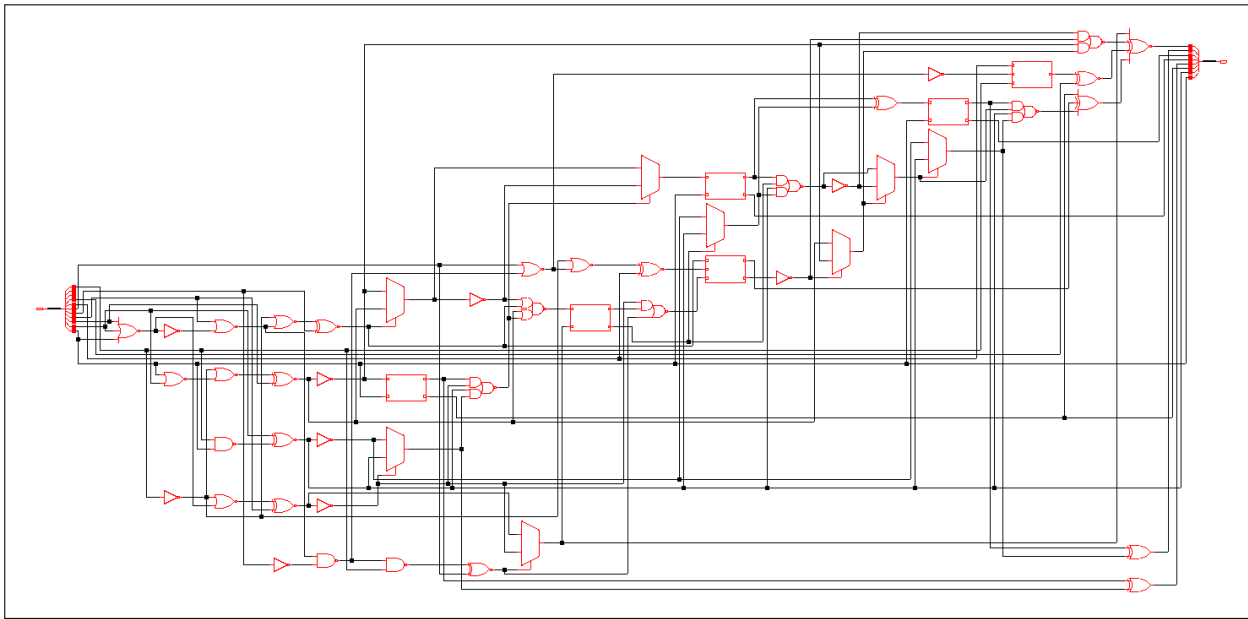


Abbildung 5.1: 12 Bit Konstantenmultiplizierer für  $\frac{\sqrt{2}}{2} = 0,70711... \simeq 0,70703 = 010110101000_2$  in Encounter

Auf Skript verweisen, mit dem ermittelt wurde, dass das die beste Annäherung an  $\frac{\sqrt{2}}{2}$  ist.

Der vollständige Gate-Report befindet sich in Abschnitt 8.2 auf Seite 29

## 5.3 Entwickeln der 2D-DFT

## 5.4 Direkte Weiterverarbeitung der Zwischenergebnisse

Um die Anzahl an Gattern und somit den Flächenbedarf zu reduzieren ist es das Ziel, die Ergebnisse der 1D-DFT aus der 1. Berechnungsstufe im nächsten Schritt direkt als Eingangswerte für die 2D-DFT zu verwenden. Auf diese Weise würden  $64 \cdot 2 \cdot 12 \text{ Bit} = 1536 \text{ Bit} = 1,5 \text{ kBit} = 192 \text{ Byte}$  an Speicher eingespart werden.

### 5.4.1 Matrizenmultiplikation

Um die nachfolgenden Abschnitte besser erörtern zu können, soll zunächst die Matrizenmultiplikation besprochen werden. Wie in Abbildung 5.1 verdeutlicht, wird Element $(i, j)$  der Ergebnismatrix dadurch berechnet, dass die Elemente $(i, k)$  einer Zeile der 1. Matrix mit den Elementn $(k, j)$  aus der zweiten Matrix multipliziert und die Werte aufsummiert werden.  $i$  und  $j$  sind für die Berechnung eines Elements konstant, während  $k$  über alle Elemente einer Zeile bzw. Spalte läuft.

$$\begin{bmatrix} \text{row } i \\ \vdots \\ \text{row } i \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \text{col } k \\ \vdots \end{bmatrix} = \begin{bmatrix} \text{row } i \\ \vdots \\ \text{row } i \end{bmatrix} \quad (5.1)$$

### 5.4.2 Veranschaulichung 2D-DFT als Matrizenmultiplikation

Mathematisch wird die 2D-DFT als

$$F_2 = W \cdot S \cdot W \quad (5.2)$$

$$= F_1 \cdot W \quad (5.3)$$

beschrieben. Es wird also erst die 1D-DFT berechnet und die sich daraus ergebende Matrix  $F_1$  (Abb. 5.4) 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 5.4 und 5.5.

$$W \cdot S = F_1 \quad (5.4)$$

$$F_1 \cdot W = F_2 \quad (5.5)$$

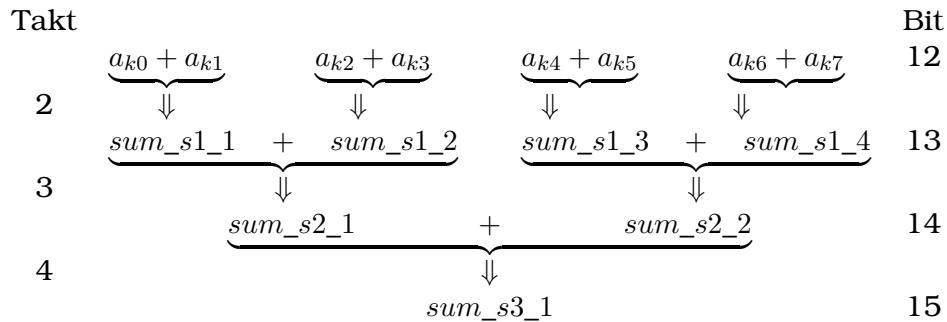
## 5.5 Optimieren der 8x8-DFT

Aus der anfänglichen Implementation bei der alle Werte einer Berechnung die entweder mit  $+\frac{\sqrt{2}}{2}$  oder  $-\frac{\sqrt{2}}{2}$  multipliziert werden müssen einzeln berechnet werden, wird sinngemäß der gemeinsame Faktor ausgeklammert, sodass nur noch jeweils eine Multiplikation erforderlich ist.

## 5.6 Ungleiche Bitbreiten bei gerader / ungerader Zeile

Berechnung ungerader Zeilen am Beispiel der ersten:

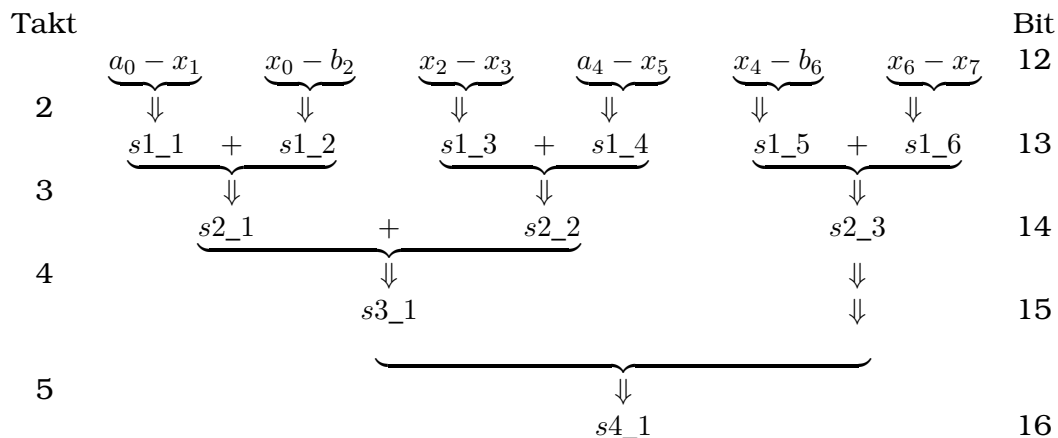
$$a_{k0} + a_{k1} + a_{k2} + a_{k3} + a_{k4} + a_{k5} + a_{k6} + a_{k7}$$



⇒ 3 Takte, 1. und 5. Leerlauf

Berechnung gerader Zeilen am Beispiel der zweiten:

$$a_0 - x_1 + x_0 - b_2 + x_2 - x_3 + a_4 - x_5 + x_4 - b_6 + x_6 - x_7$$



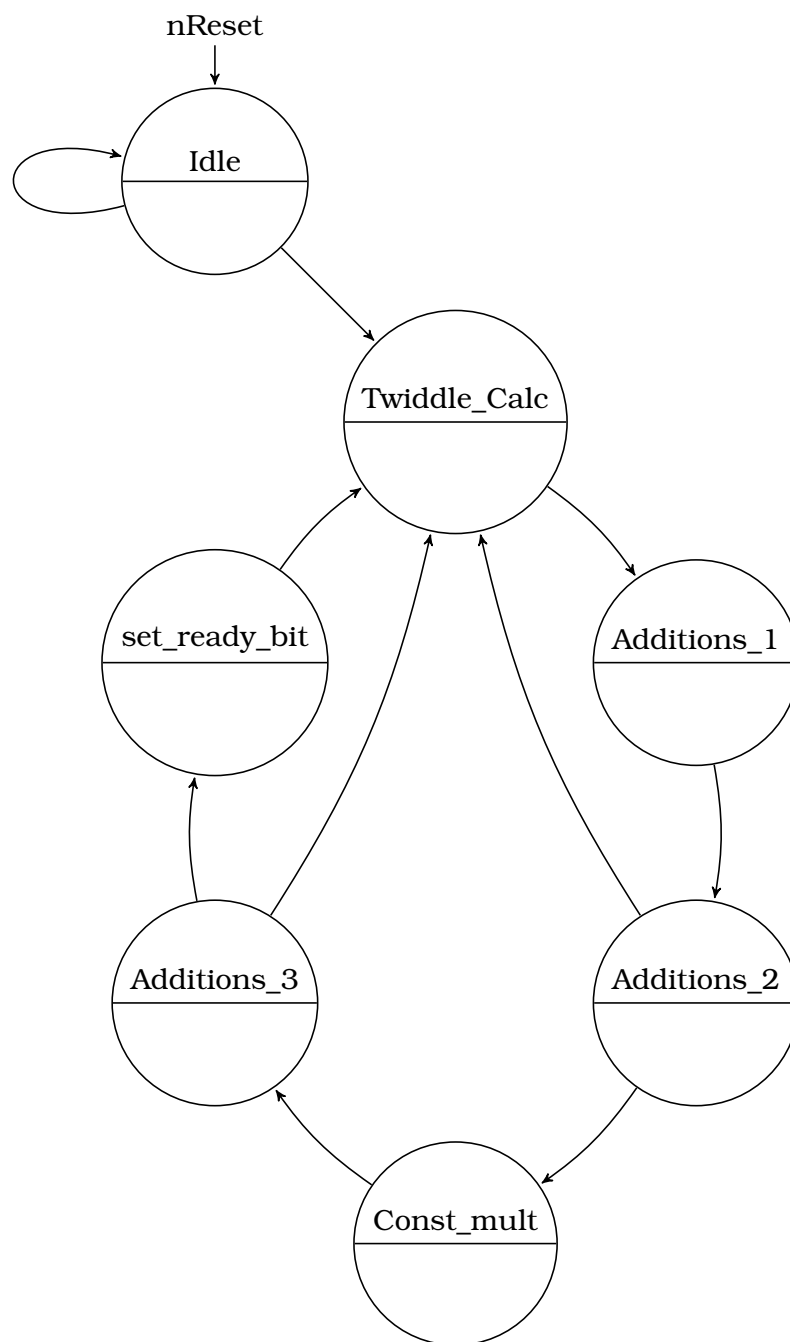
⇒ 5 Takte, 1. Multiplikationen, 2.-5. Additionen

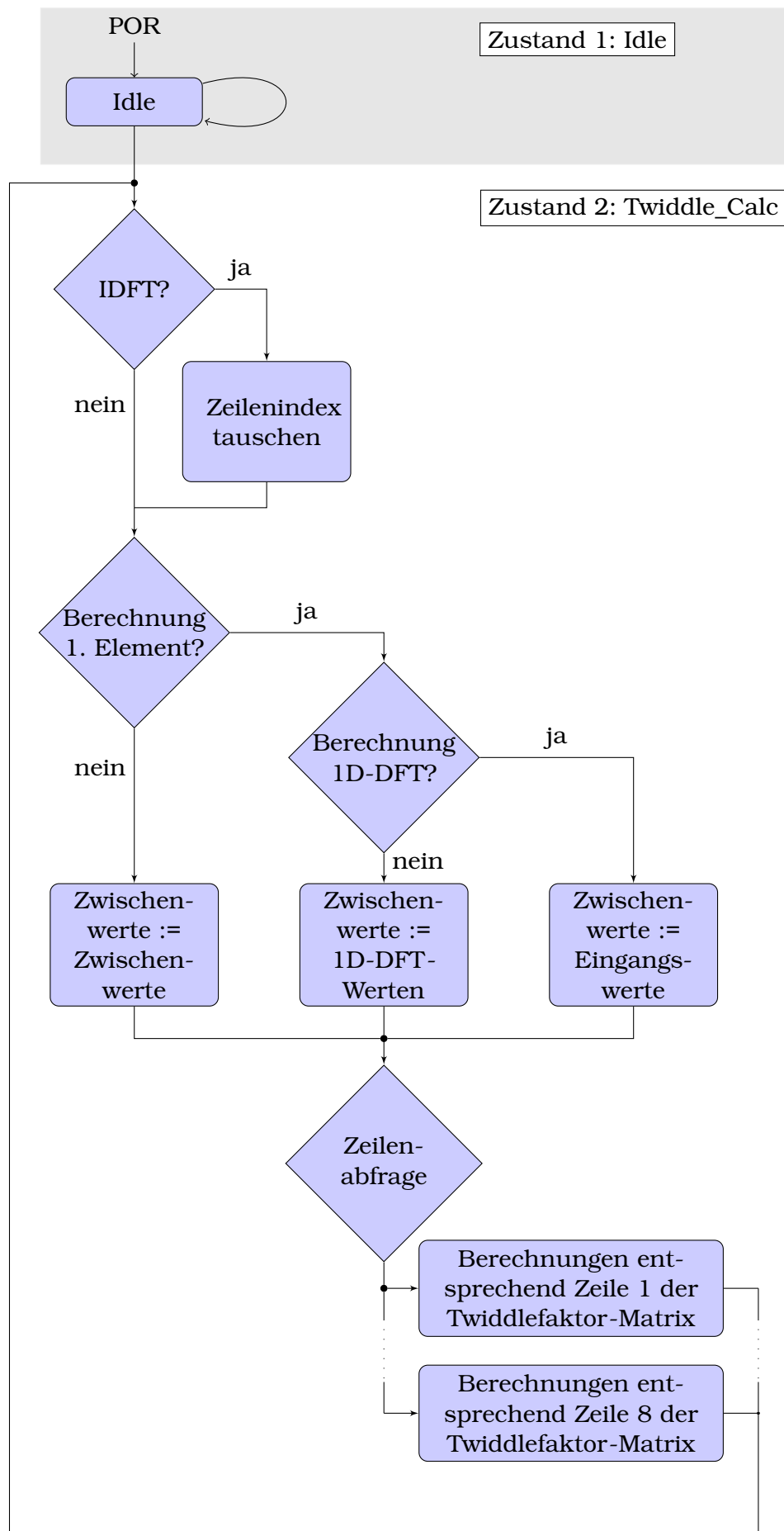
## 5.7 Struktogramm

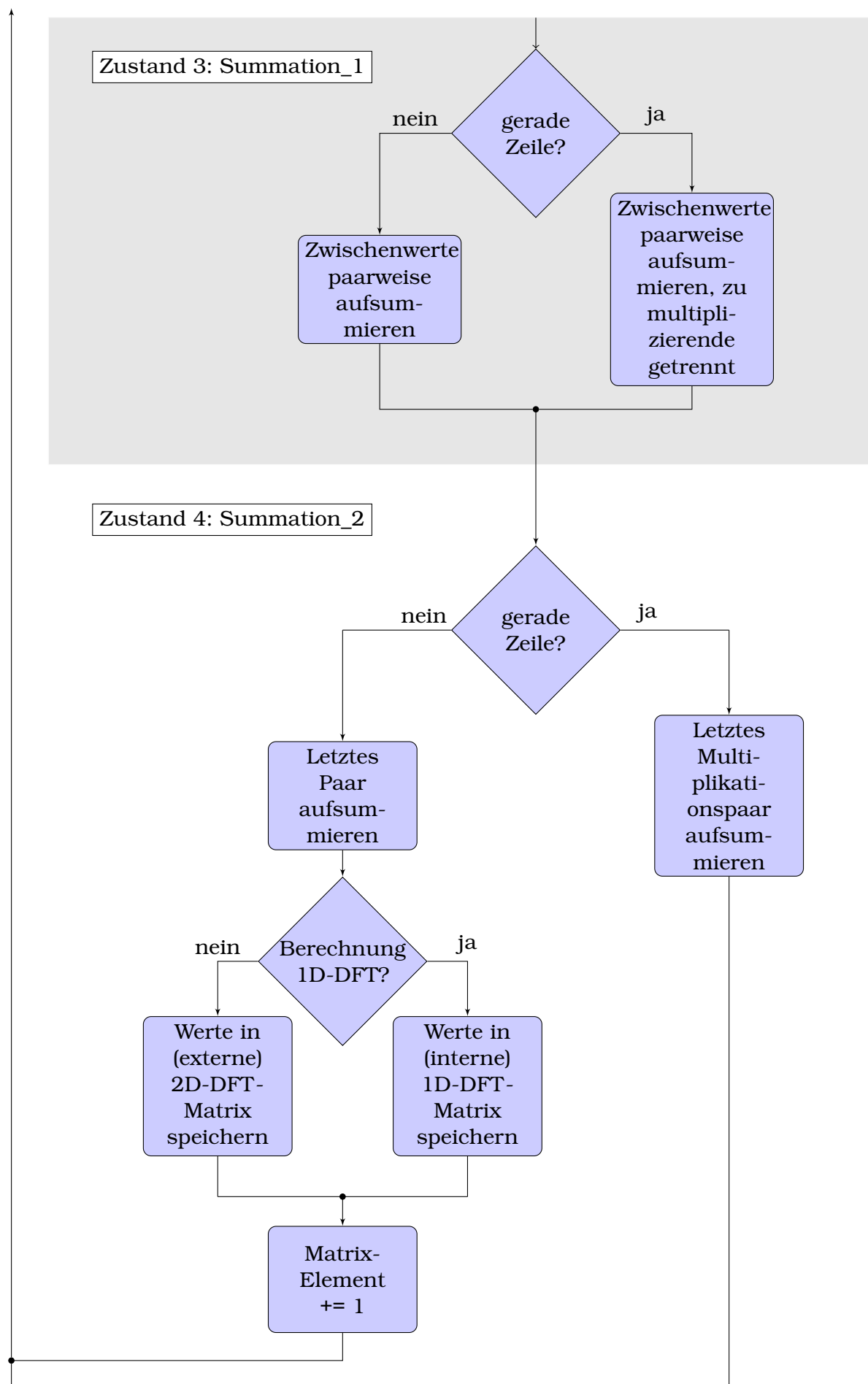
## 2D-DFT

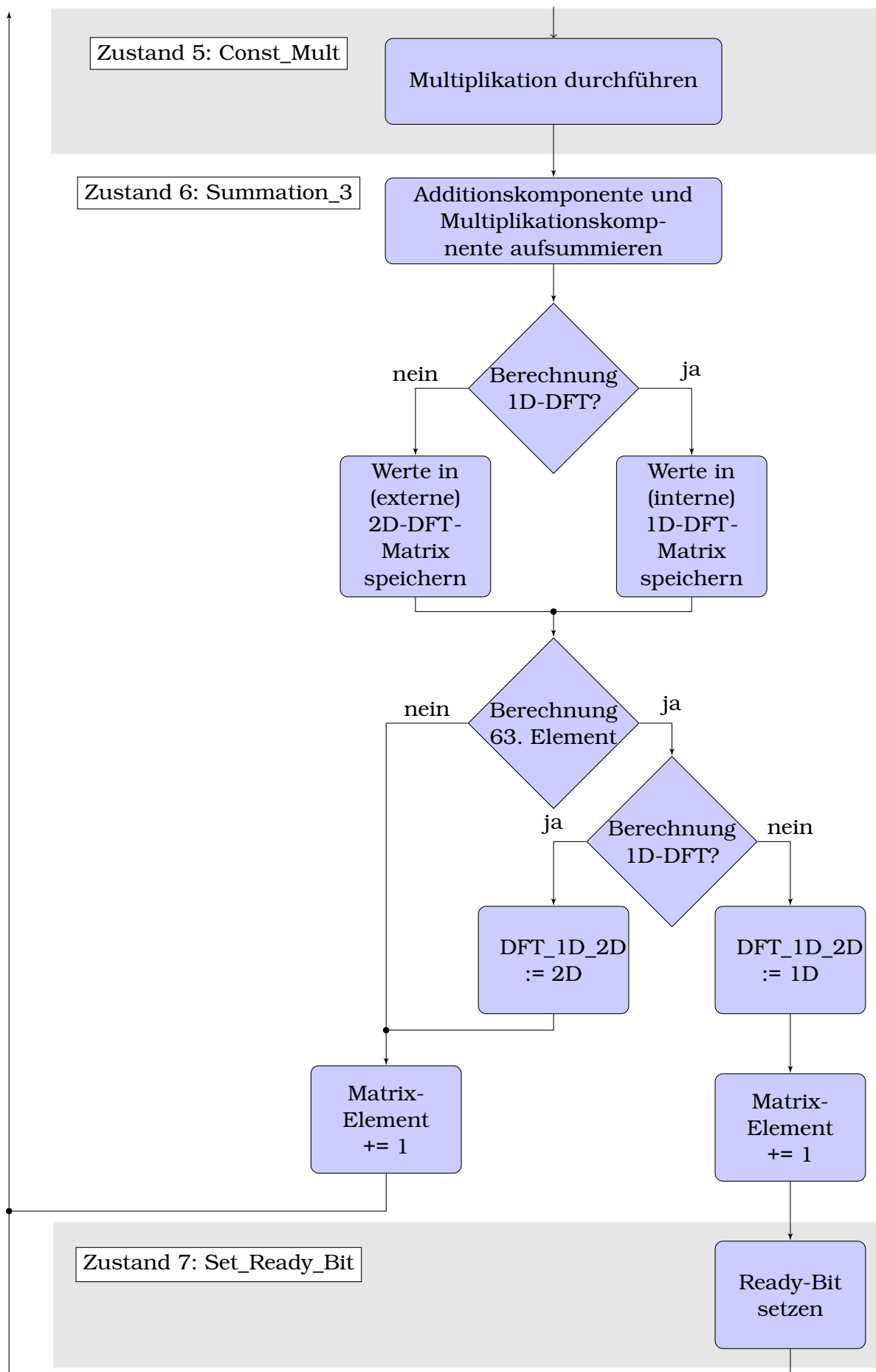
Berechnung des 1. Elements der Ergebnismatrix?								F
T								
1. Durchlauf? (1. Durchlauf : 1D-DFT, 2. Durchlauf : 2D-DFT)								F
T								
interne Matrix := Eingangswerte-Matrix;						interne Matrix := 1D-DFT-Matrix;		
Zeile der Twiddlefaktor-Matrix								
1	2	3	4	5	6	7	8	
Additionen gemäß der 1. Zeile der Twiddlefaktor-Matrix für das n-te Element der Ergebnismatrix	Subtraktionen gemäß 2. Zeile ...	Subtraktionen gemäß 3. Zeile ...	... 4. Zeile ...	... 5. Zeile ...	... 6. Zeile ...	... 7. Zeile ...	... 8. Zeile ...	
ungerade Zeile der Twiddlefaktor-Matrix?								F
T								
Die 4 Zwischenergebnisse aufsummieren				Die 6 Zwischenwerte getrennt nach denen, die später noch mit Wurzel(2)/2 multipliziert werden müssen und denen, die es nicht müssen, aufsummieren.				
1. Durchlauf?				T				F
"final" aufsummieren und Ergebnis der internen 1D-DFT-Matrix zuweisen				final aufsummieren und Ergebnis der externen Ergebnis-Matrix zuweisen.				
				Multiplikation mit Wurzel(2)/2				
				1. Durchlauf?				
				T				F
				"final" aufsummieren und Ergebnis der internen 1D-DFT-Matrix zuweisen				
				final aufsummieren und Ergebnis der externen Ergebnis-Matrix zuweisen.				
element = 63? (Berechnung des 64. Elements?)								F
T								
1. Durchlauf?								F
T								
						Ergebnis steht bereit		
Durchlauf-Bit wird getoggelt								
Matrix-Element := Matrix-Element + 1; (bezogen auf die Ergebnismatrix, läuft von 0 bis 63 und hat dann einen gewollten Überlauf (6 Bit))								

## 5.8 Automatengraf











## 6 Evaluation

### 6.1 Simulation

#### 6.1.1 NC Sim - positive Zahlendarstellung

#### 6.1.2 Anzahl benötigter Takte

Anhand der Simulation kann die Anzahl der vorausgesagten benötigten Takte verifiziert werden.

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 6.1 bei 340 ns, beginnt die Berechnung der 2D-DFT. Beendet ist 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  $(40 \text{ ns})^{-1}$  (siehe 8.16) 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.

#### 6.1.3 Akkumulation von Fehlern

Durch die Begrenzung der Bitbreite ist es nötig nach jeder Addition den Wert zu halbieren. Hierbei steigt die Abweichung gegenüber einer verlustfreien Berechnung immer dann, wenn das letzte eine 1 ist. Im Mittel ist dies bei der Hälfte der Additionen der Fall. In der Hälfte aller Fälle wird also der Wert um ein halbes LSB zu viel verringert. Bei der Multiplikation sieht es nicht besser aus, hier werden aus einem 24 Bit Vektor ebenfalls nur 12 übernommen. Da für die Berechnung einer Zahl der 1D-DFFT je nach Zeile entweder 8 oder 12 Werte akkumuliert sowie 0 bis 4 Werte multipliziert werden, für die 2D-DFT entsprechend doppelt so viele, akkumulieren sich zwangsläufig Fehler. Bei 12 Bit Eingangswerten wäre ein 47 Bit Ausgangsvektor nötig, um dies vollständig zu vermeiden. Dies ist jedoch aus u.a. Platzgründen nicht umsetzbar.  $\Rightarrow$  Anhand eines Simulationsbeispiels zeigen, dass die mit VHDL berechneten Werte immer kleiner als die in Matlab berechneten sind.

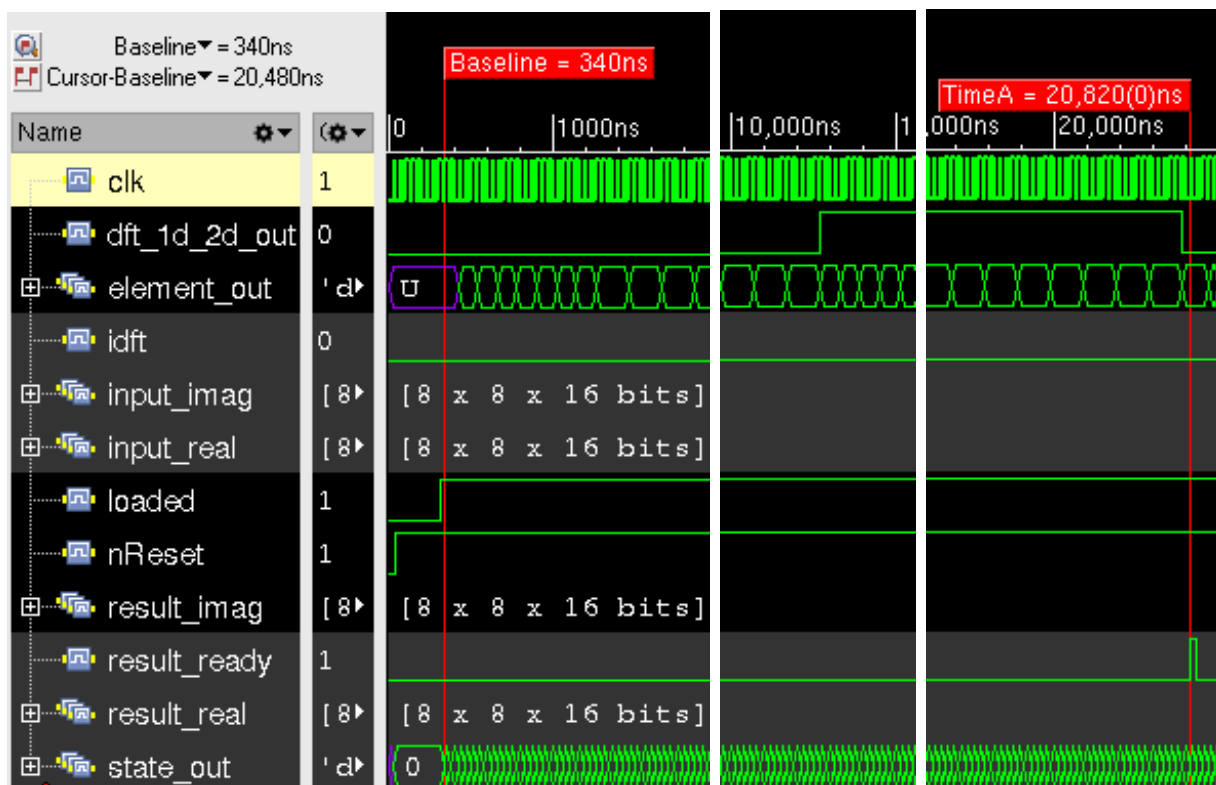


Abbildung 6.1: Simulations der 2D-DFT mit NC Launch

## 6.2 Testumgebung

### 6.2.1 1D-DFT, Integer

### 6.2.2 2D-DFT, Integer

### 6.2.3 2D-DFT, Q-Format

### 6.2.4 Struktogramm des Testablaufs

### 6.2.5 Reale Eingangswerte

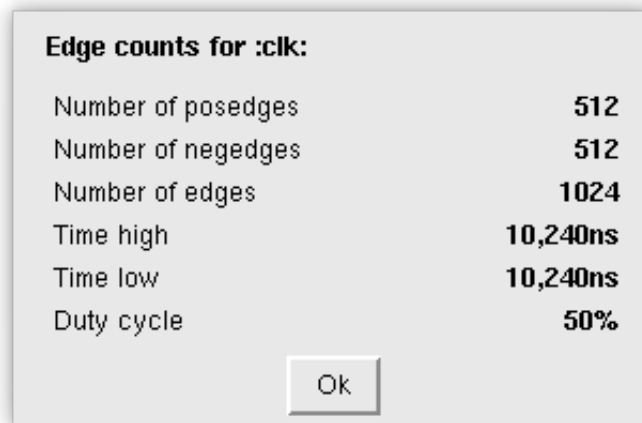
## 6.3 Anzahl Standardzellen

### 6.3.1 1D / 2D

### 6.3.2 3 Lagen / 4 Lagen

## 6.4 Visualisierung der Netzliste

## 6.5 Floorplan, Pading



<b>Edge counts for :clk:</b>	
Number of posedges	<b>512</b>
Number of negedges	<b>512</b>
Number of edges	<b>1024</b>
Time high	<b>10,240ns</b>
Time low	<b>10,240ns</b>
Duty cycle	<b>50%</b>

Ok

Abbildung 6.2: Edge Count für eine 2D-DFT

## **7 Schlussfolgerungen**

### **7.1 Zusammenfassung**

### **7.2 Bewertung und Fazit**

### **7.3 Ausblick**

# Abbildungsverzeichnis

3.1	Einheitskreis, Zusammensetzung des komplexen Zeigers aus Sinus und Kosinus . . . . .	4
4.1	Einheitskreis mit relevanten Werten . . . . .	9
4.2	Twiddlefaktoren der $8 \times 8$ -Matrix, aufgeteilt auf die Laufindexe . . . . .	9
4.3	Matrizen-Darstellung der Twiddlefaktoren aufgeteilt nach Real- und Imaginärteil . . . . .	9
5.1	12 Bit Konstantenmultiplizierer für $\frac{\sqrt{2}}{2} = 0,70711... \simeq 0,70703 = 010110101000_2$ in Encounter . . . . .	13
6.1	Simulations der 2D-DFT mit NC Launch . . . . .	22
6.2	Edge Count für eine 2D-DFT . . . . .	23

## Tabellenverzeichnis

4.1	Bewertung der Twiddlefaktor-Matrizen . . . . .	7
5.1	Vergleich Konstanten- mit regulärem Multiplizierer . . . . .	12

## Literatur

- [1] M. Krey, „Systemarchitektur und Signalverarbeitung für die Diagnose von magnetischen ABS-Sensoren“, *test*, 2015.

## 8 Anhang

### 8.1 Skript zur Bewertung von 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, +0.5, +1
6 %%           Es wird ein Verhaeltnis aus trivialen und nicht trivialen Werten
7 %%           erstellt.
8 %% Argumente: N (Groesse der NxN DFT-Matrix)
9 %% Author:    Thomas Lattmann
10 %% Datum:     17.10.2017
11 %% Version:   1.0

13 function dft_bewertung(N)    % N : Groesse der Matrix (NxN)

15     % Twiddlefaktor-Matrix erzeugen
16     tw_cmplx = exp(-i*2*pi*[0:N-1]'*[0:N-1]/N);
17
18     % Matrix nach Im und Re trennen und Werte runden
19     tw_real = round(real(tw_cmplx)*100000)/100000;
20     tw_imag = round(imag(tw_cmplx)*100000)/100000;
21
22     % Werte kleiner 0,000001 auf 0 setzen (arithmetische Ungenauigkeiten)
23     tw_real(abs(tw_real) < 0.000001) = 0;
24     tw_imag(abs(tw_imag) < 0.000001) = 0;
25
26     %% Anzahl verschiedener Werte ermitteln
27     different_nums_real = unique(tw_real);
28     different_nums_imag = unique(tw_imag);
29
30     % Jeweils die Menge der verschiedenen Werte ermitteln (hier Re)
31     num_count_real = zeros(1, length(different_nums_real));
32     for k = 1:length(different_nums_real)
33         for n = 1:N
34             for m = 1:N
35                 if different_nums_real(k) == tw_real(m,n)
36                     num_count_real(k) = num_count_real(k) + 1;
37                 end
38             end
39         end
40     end
41
42     % Jeweils die Anzahl der verschiedenen Werte ermitteln (hier Im)
43     num_count_imag = zeros(1, length(different_nums_imag));
44     for k = 1:length(different_nums_imag)
45         for n = 1:N
46             for m = 1:N
47                 if different_nums_imag(k) == tw_imag(m,n)
```



```

49         num_count_imag(k) = num_count_imag(k) + 1;
50     end
51 end
52 end
53 end
54
55 % nicht triviale Werte der reellen Matrix zaehlen
56 nontrivial_nums_real = 0;
57 for k = 1:length(different_nums_real)
58     if abs(different_nums_real(k)) != 1
59         if abs(different_nums_real(k)) != 0.5
60             if different_nums_real(k) != 0
61                 nontrivial_nums_real = nontrivial_nums_real + num_count_real(k);
62             end
63         end
64     end
65 end
66 end
67
68 % nicht triviale Werte der imaginaeren Matrix zaehlen
69 nontrivial_nums_imag = 0;
70 for k = 1:length(different_nums_imag)
71     if abs(different_nums_imag(k)) != 1
72         if abs(different_nums_imag(k)) != 0.5
73             if different_nums_imag(k) != 0
74                 nontrivial_nums_imag = nontrivial_nums_imag + num_count_imag(k);
75             end
76         end
77     end
78 end
79
80 nums_of_each_matrix = N*N;
81
82 trivial_nums_real = N*N - nontrivial_nums_real
83 trivial_nums_imag = N*N - nontrivial_nums_imag
84
85 nontrivial_nums_real
86 nontrivial_nums_imag
87
88 trivial_nums_total = trivial_nums_real + trivial_nums_imag
89 nontrivial_nums_total = nontrivial_nums_real + nontrivial_nums_imag
90
91 v = trivial_nums_total/nontrivial_nums_total
92
93 end

```

Listing 8.1: Octave-Skript zur Bewertung unterschiedlicher Twiddlefaktormatrizen

## 8.2 Gate-Report des 12 Bit Konstantenmultiplizierers

```

1 rc:/> report gates
=====
3  Generated by:      Encounter(R) RTL Compiler RC14.25 - v14.20-s046_1
4  Generated on:      May 30 2017 03:29:41 pm
5  Module:            multiplier
6  Technology library: c35_CORELIB_TYP 3.02

```

```

7  Operating conditions:  _nominal_ (balanced_tree)
  Wireload mode:        enclosed
9  Area mode:           timing library
=====
11
13  Gate      Instances    Area      Library
-----
15  ADD21          5      728.000   c35_CORELIB_TYP
16  AOI210         2      145.600   c35_CORELIB_TYP
17  AOI220        18     1638.000   c35_CORELIB_TYP
18  CLKIN0         6      218.400   c35_CORELIB_TYP
19  IMUX20        38     3458.000   c35_CORELIB_TYP
20  INVO          27      982.800   c35_CORELIB_TYP
21  NAND20        12      655.200   c35_CORELIB_TYP
22  NOR20         8      436.800   c35_CORELIB_TYP
23  OAI220         6      546.000   c35_CORELIB_TYP
24  XNR20         15     1638.000   c35_CORELIB_TYP
25  XNR30         6     1201.200   c35_CORELIB_TYP
26  XNR31         3      600.600   c35_CORELIB_TYP
27  XOR20         5      637.000   c35_CORELIB_TYP
-----
29  total          151   12885.600
30
31
33  Type      Instances    Area      Area %
-----
35  inverter         33   1201.200     9.3
36  logic          118  11684.400    90.7
37
38  total          151  12885.600   100.0
39
rc:/>

```

Listing 8.2: RC Gate-Report

### 8.3 Twiddlefaktormatrix im S1Q10-Format

```

1  %% Dateiname:      twiddle2file.m
  %% Funktion:       Erzeugt eine Datei mit den binären komplexen
3  %%                Twiddlefaktoren
  %% Argumente:      N (Größe der NxN DFT-Matrix)
5  %% Aufbau der Datei: Wie die Matrix, enthält Realteil und Imaginärteil.
  %%                Alle Werte sind wie im Beispiel durch Leerzeichen getrennt:
7  %%                Re{W(1,1)} Im{W(1,1)} Re{W(1,2)} Im{W(1,2)}
  %%                Re{W(2,1)} Im{W(2,1)} Re{W(2,2)} Im{W(2,2)}
9  %% Abhängigkeiten: (1) twiddle_coefficients.m
  %%                (2) dec_to_slq10.m
11 %%                (3) bit_vector2integer.m
  %%                (4) zweier_komplement.m
13 %% Author:        Thomas Lattmann
  %% Datum:          02.11.17
15 %% Version:       1.0
17 function twiddle2file(N)

```

```

19 % Dezimale Twiddlefaktormatrix erstellen
W_dec = twiddle_coefficients(N);
21 W_dec_real = real(W_dec);
W_dec_imag = imag(W_dec);
23
W_bin_int_real = zeros(size(W_dec_real));
25 W_bin_int_imag = zeros(size(W_dec_imag));
27
for m = 1:N
    for n = 1:N
29         bit_vector = dec_to_slq10(W_dec_real(m,n));
W_bin_int_real(m,n) = bit_vector2integer(bit_vector);
31
        bit_vector = dec_to_slq10(W_dec_imag(m,n));
33         W_bin_int_imag(m,n) = bit_vector2integer(bit_vector);
    end
35 end
37 fid=fopen('Twiddle_slq10_komplex.txt','w+');
39
for m=1:N
    for n=1:N
41         fprintf(fid, '%012d ', W_bin_int_real(m,n));
fprintf(fid, '%012d ', W_bin_int_imag(m,n));
43         if n < N
            fprintf(fid, ' ');
45         end
    end
47     if m < N
        fprintf(fid, '\n');
49     end
end
51
fclose(fid);
53
end

```

Listing 8.3: 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);

```

```

20 W_imag = imag(W);
21 W_real(abs(W_real)<00000.1) = 0;
22 W_imag(abs(W_imag)<00000.1) = 0;
23 W = W_real + i*W_imag;
24 end

```

Listing 8.4: Erzeugen der Twiddlefaktormatrix

```

%% Dateiname:      dec_to_slq10.m
2 %% Funktion:      Konvertiert eine Dezimalzahl in das binaere SlQ10-Format
%% Argumente:      Dezimalzahl im Bereich von  $-2 \dots +2 - 1/2^{10}$ 
4 %% Abhaengigkeiten: (1) zweier_komplement.m
%% Author:         Thomas Lattmann
6 %% Datum:         02.11.17
%% Version:        1.0
8
function bit_vector = dec_to_slq10(val)
10
11 bit_width=12;
12 bit_vector=zeros(1,bit_width);
13 dec_temp=0;
14 val_abs=abs(val);
15 val_int=floor(val_abs);
16 val_frac=val_abs-val_int;
17
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 slq11-Format dargestellt werden.')
20 elseif val < -2
    disp('Diese Zahl kann nicht im slq11-Format dargestellt werden.')
22 else
23
24     % Vorkommastellen
25     if abs(val) >= 1
26         bit_vector(2) = 1;
27         if val == -2
28             bit_vector(1) = 1;
29         end
30     end
31
32     % Nachkommastellen
33     for k = 1:bit_width-2
34         % berechnen der Differenz des Twiddlefaktors und des derzeitigen Wertes der
35         Binaerzahl
36         d = val_frac - dec_temp;
37         if d >= 1/2^k
38             bit_vector(k+2) = 1;
39             dec_temp = dec_temp+1/2^k;
40         end
41     end
42
43     % 2er-Komplement bilden, falls val negativ
44     if val < 0
45         bit_vector=zweier_komplement(bit_vector);
46     end
end

```

Listing 8.5: Dezimalzahl nach S1Q10 konvertieren

```

1 %% Dateiname: zweier_komplement.m
2 %% Funktion: Bilden des 2er-Komplements eines "Bit"-Vektors
3 %% Argumente: Vektor aus Nullen und Einsen
4 %% Author: Thomas Lattmann
5 %% Datum: 02.11.17
6 %% Version: 1.0
7
8 function bit_vector = zweier_komplement(bit_vector)
9     bit_width=length(bit_vector);
10
11     for j = 1:bit_width
12         bit_vector(j) = not(bit_vector(j));
13     end
14     bit_vector(bit_width) = bit_vector(bit_width) + 1;
15     for j = 1:bit_width-1
16         if bit_vector(bit_width - j + 1) == 2
17             bit_vector(bit_width - j + 1) = 0;
18             bit_vector(bit_width - j) = bit_vector(bit_width - j) + 1;
19         end
20     end
21 end

```

Listing 8.6: Bildung des 2er-Komplements

```

1 %% Dateiname: bit_vector2integer.m
2 %% Funktion: Wandelt einen Vektor von Zahlen in eine einzelne Zahl (Integer)
3 %% Beispiel: [0 1 1 0 0 1] => 11001
4 %% Um fuhrende Nullen zu erhalten muss z.B. printf('%06d', Integer)
5 %% genutzt werden. Hierbei wird vorne mit Nullen aufgefuellt, wenn
6 %% 'Integer' weniger als 6 stellen hat.
7 %% Argumente: Vektor (aus Nullen und Einsen)
8 %% Author: Thomas Lattmann
9 %% Datum: 02.11.17
10 %% Version: 1.0
11
12 function bin_int = bit_vector2integer(bit_vector)
13
14     bin_int=0;
15     bit_width=length(bit_vector);
16
17     % Konvertierung von Vektor nach Integer
18     for l = 1:bit_width
19         bin_int = bin_int + bit_vector(bit_width - l + 1)*10^(l-1);
20     end
21 end

```

Listing 8.7: Binär-Vektor in Binär-Integer umwandeln

```

1 %% Dateiname: slq10_to_dec.m
2 %% Funktion: Konvertiert eine binaere Zahl im S1Q10-Format als Dezimalzahl
3 %% Argumente: Vektor aus Nullen und Einsen
4 %% Author: Thomas Lattmann
5 %% Datum: 02.11.17

```

```

6 %% Version:    1.0
8 function dec = slq10_to_dec(bit_vector)
10 % Dezimalzahl aus slq10 Binaerzahl berechnen
12 bit_width=length(bit_vector);
14 dec = 0;
16 if bit_vector(1) == 1
18     dec = -2;
20     if bit_vector(2) == 1
22         dec = -1;
24     end
26 elseif bit_vector(2) == 1
28     dec = 1;
30 end
32 for n = 3:bit_width
34     if bit_vector(n) == 1
36         dec = dec + 1/2^(n-2);
38     end
40 end
42 end

```

Listing 8.8: Kontroll-Skript für S1Q10 nach Dezimal

## 8.4 Ausmultiplizierte Matrizen

## 8.5 Ausmultiplizieren der 8x8 DFT

$$\begin{bmatrix}
 1+j0 & 1+j0 & 1+j0 & 1+j0 & 1+j0 & 1+j0 & 1+j0 & 1+j0 \\
 1+j0 & \frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2} & 0+j1 & -\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2} & -1+j0 & -\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2} & 0-j1 & \frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2} \\
 1+j0 & 0+j1 & -1+j0 & 0-j1 & 1+j0 & 0+j1 & -1+j0 & 0-j1 \\
 1+j0 & -\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2} & 0-j1 & \frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2} & -1+j0 & \frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2} & 0+j1 & -\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2} \\
 1+j0 & -1+j0 & 1+j0 & -1+j0 & 1+j0 & -1+j0 & 1+j0 & -1+j0 \\
 1+j0 & -\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2} & 0+j1 & \frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2} & -1+j0 & \frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2} & 0-j1 & -\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2} \\
 1+j0 & 0-j1 & -1+j0 & 0+j1 & 1+j0 & 0-j1 & -1+j0 & 0+j1 \\
 1+j0 & \frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2} & 0-j1 & -\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2} & -1+j0 & -\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2} & 0+j1 & \frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}
 \end{bmatrix}$$

$$i = \text{const.} \downarrow \begin{bmatrix} a_{00} + jb_{00} & a_{01} + jb_{01} & a_{02} + jb_{02} & a_{03} + jb_{03} & a_{04} + jb_{04} & a_{05} + jb_{05} & a_{06} + jb_{06} & a_{07} + jb_{07} \\ a_{10} + jb_{10} & a_{11} + jb_{11} & a_{12} + jb_{12} & a_{13} + jb_{13} & a_{14} + jb_{14} & a_{15} + jb_{15} & a_{16} + jb_{16} & a_{17} + jb_{17} \\ a_{20} + jb_{20} & a_{21} + jb_{21} & a_{22} + jb_{22} & a_{23} + jb_{23} & a_{24} + jb_{24} & a_{25} + jb_{25} & a_{26} + jb_{26} & a_{27} + jb_{27} \\ a_{30} + jb_{30} & a_{31} + jb_{31} & a_{32} + jb_{32} & a_{33} + jb_{33} & a_{34} + jb_{34} & a_{35} + jb_{35} & a_{36} + jb_{36} & a_{37} + jb_{37} \\ a_{40} + jb_{40} & a_{41} + jb_{41} & a_{42} + jb_{42} & a_{43} + jb_{43} & a_{44} + jb_{44} & a_{45} + jb_{45} & a_{46} + jb_{46} & a_{47} + jb_{47} \\ a_{50} + jb_{50} & a_{51} + jb_{51} & a_{52} + jb_{52} & a_{53} + jb_{53} & a_{54} + jb_{54} & a_{55} + jb_{55} & a_{56} + jb_{56} & a_{57} + jb_{57} \\ a_{60} + jb_{60} & a_{61} + jb_{61} & a_{62} + jb_{62} & a_{63} + jb_{63} & a_{64} + jb_{64} & a_{65} + jb_{65} & a_{66} + jb_{66} & a_{67} + jb_{67} \\ a_{70} + jb_{70} & a_{71} + jb_{71} & a_{72} + jb_{72} & a_{73} + jb_{73} & a_{74} + jb_{74} & a_{75} + jb_{75} & a_{76} + jb_{76} & a_{77} + jb_{77} \end{bmatrix}$$

### 1. Zeile:

$$(1 + j0) \cdot (a_{0i} + jb_{0i}) + (1 + j0) \cdot (a_{1i} + jb_{1i}) + (1 + j0) \cdot (a_{2i} + jb_{2i}) + (1 + j0) \cdot (a_{3i} + jb_{3i}) + (1 + j0) \cdot (a_{4i} + jb_{4i}) + (1 + j0) \cdot (a_{5i} + jb_{5i}) + (1 + j0) \cdot (a_{6i} + jb_{6i}) + (1 + j0) \cdot (a_{7i} + jb_{7i})$$

$$= a_{0i} + jb_{0i} + a_{1i} + jb_{1i} + a_{2i} + jb_{2i} + a_{3i} + jb_{3i} + a_{4i} + jb_{4i} + a_{5i} + jb_{5i} + a_{6i} + jb_{6i} + a_{7i} + jb_{7i}$$

$$\Rightarrow \Re_{0i} = a_{0i} + a_{1i} + a_{2i} + a_{3i} + a_{4i} + a_{5i} + a_{6i} + a_{7i}$$

$$\Rightarrow \Im_{0i} = b_{0i} + b_{1i} + b_{2i} + b_{3i} + b_{4i} + b_{5i} + b_{6i} + b_{7i}$$

### 2. Zeile:

$$(1 + j0) \cdot (a_{0i} + jb_{0i}) + (\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}) \cdot (a_{1i} + jb_{1i}) + (0 + j1) \cdot (a_{2i} + jb_{2i}) + (-\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}) \cdot (a_{3i} + jb_{3i}) + (-1 + j0) \cdot (a_{4i} + jb_{4i}) + (-\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}) \cdot (a_{5i} + jb_{5i}) + (0 - j1) \cdot (a_{6i} + jb_{6i}) + (\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}) \cdot (a_{7i} + jb_{7i})$$

$$(1 + j0) \cdot (a_{0i} + jb_{0i}) = a_{0i} + jb_{0i}$$

$$\rightarrow \Re = a_{0i}, \quad \Im = b_{0i}$$

$$(\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}) \cdot (a_{1i} + jb_{1i}) = \frac{\sqrt{2}}{2} \cdot a_{1i} + j\frac{\sqrt{2}}{2} \cdot a_{1i} + j\frac{\sqrt{2}}{2} \cdot b_{1i} - \frac{\sqrt{2}}{2} \cdot b_{1i}$$

$$\rightarrow \Re = \frac{\sqrt{2}}{2} \cdot a_{1i} - \frac{\sqrt{2}}{2} \cdot b_{1i}, \quad \Im = \frac{\sqrt{2}}{2} \cdot a_{1i} + \frac{\sqrt{2}}{2} \cdot b_{1i}$$

$$(0 + j1) \cdot (a_{2i} + jb_{2i}) = -b_{2i} + ja_{2i}$$

$$\rightarrow \Re = -b_{2i}, \quad \Im = a_{2i}$$

$$(-\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}) \cdot (a_{3i} + jb_{3i}) = -\frac{\sqrt{2}}{2} \cdot a_{3i} + j\frac{\sqrt{2}}{2} \cdot a_{3i} - j\frac{\sqrt{2}}{2} \cdot b_{3i} - \frac{\sqrt{2}}{2} \cdot b_{3i}$$

$$\rightarrow \Re = -\frac{\sqrt{2}}{2} \cdot a_{3i} - \frac{\sqrt{2}}{2} \cdot b_{3i}, \quad \Im = \frac{\sqrt{2}}{2} \cdot a_{3i} - \frac{\sqrt{2}}{2} \cdot b_{3i}$$

$$\underline{(-1 + j0) \cdot (a_{4i} + jb_{4i})} = -a_{4i} - jb_{4i}$$

$$\rightarrow \Re = -a_{4i}, \quad \Im = -b_{4i}$$

$$\underline{\left(-\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}\right) \cdot (a_{5i} + jb_{5i})} = -\frac{\sqrt{2}}{2} \cdot a_{5i} - j\frac{\sqrt{2}}{2} \cdot a_{5i} - j\frac{\sqrt{2}}{2} \cdot b_{5i} + \frac{\sqrt{2}}{2} \cdot b_{5i}$$

$$\rightarrow \Re = -\frac{\sqrt{2}}{2} \cdot a_{5i} + \frac{\sqrt{2}}{2} \cdot b_{5i}, \quad \Im = -\frac{\sqrt{2}}{2} \cdot a_{5i} - \frac{\sqrt{2}}{2} \cdot b_{5i}$$

$$\underline{(0 - j1) \cdot (a_{6i} + jb_{6i})} = b_{6i} - ja_{6i}$$

$$\rightarrow \Re = b_{6i}, \quad \Im = -a_{6i}$$

$$\underline{\left(\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}\right) \cdot (a_{7i} + jb_{7i})} = \frac{\sqrt{2}}{2} \cdot a_{7i} - j\frac{\sqrt{2}}{2} \cdot a_{7i} + j\frac{\sqrt{2}}{2} \cdot b_{7i} + \frac{\sqrt{2}}{2} \cdot b_{7i}$$

$$\rightarrow \Re = \frac{\sqrt{2}}{2} \cdot a_{7i} + \frac{\sqrt{2}}{2} \cdot b_{7i}, \quad \Im = -\frac{\sqrt{2}}{2} \cdot a_{7i} + \frac{\sqrt{2}}{2} \cdot b_{7i}$$

$$\Rightarrow \Re_{1i} = a_{0i} + \frac{\sqrt{2}}{2} \cdot a_{1i} - \frac{\sqrt{2}}{2} \cdot b_{1i} - b_{2i} - \frac{\sqrt{2}}{2} \cdot a_{3i} - \frac{\sqrt{2}}{2} \cdot b_{3i} - a_{4i} - \frac{\sqrt{2}}{2} \cdot a_{5i} + \frac{\sqrt{2}}{2} \cdot b_{5i} + b_{6i} + \frac{\sqrt{2}}{2} \cdot a_{7i} + \frac{\sqrt{2}}{2} \cdot b_{7i}$$

$$\Rightarrow \Im_{1i} = b_{0i} + \frac{\sqrt{2}}{2} \cdot a_{1i} + \frac{\sqrt{2}}{2} \cdot b_{1i} + a_{2i} + \frac{\sqrt{2}}{2} \cdot a_{3i} - \frac{\sqrt{2}}{2} \cdot b_{3i} - b_{4i} - \frac{\sqrt{2}}{2} \cdot a_{5i} - \frac{\sqrt{2}}{2} \cdot b_{5i} - a_{6i} - \frac{\sqrt{2}}{2} \cdot a_{7i} + \frac{\sqrt{2}}{2} \cdot b_{7i}$$

### 3. Zeile:

$$\underline{(1 + j0) \cdot (a_{0i} + jb_{0i})} + \underline{(0 + j1) \cdot (a_{1i} + jb_{1i})} + \underline{(-1 + j0) \cdot (a_{2i} + jb_{2i})} + \underline{(0 - j1) \cdot (a_{3i} + jb_{3i})} +$$

$$\underline{(1 + j0) \cdot (a_{4i} + jb_{4i})} + \underline{(0 + j1) \cdot (a_{5i} + jb_{5i})} + \underline{(-1 + j0) \cdot (a_{6i} + jb_{6i})} + \underline{(0 - j1) \cdot (a_{7i} + jb_{7i})}$$

$$\underline{(1 + j0) \cdot (a_{0i} + jb_{0i})} = a_{0i} + jb_{0i}$$

$$\rightarrow \Re = a_{0i}, \quad \Im = b_{0i}$$

$$\underline{(0 + j1) \cdot (a_{1i} + jb_{1i})} = -b_{1i} + ja_{1i}$$

$$\rightarrow \Re = -b_{1i}, \quad \Im = a_{1i}$$

$$\underline{(-1 + j0) \cdot (a_{2i} + jb_{2i})} = -a_{2i} - jb_{2i}$$

$$\rightarrow \Re = -a_{2i}, \quad \Im = -b_{2i}$$

$$\underline{(0 - j1) \cdot (a_{3i} + jb_{3i})} = b_{3i} - ja_{3i}$$

$$\rightarrow \Re = b_{3i}, \quad \Im = -a_{3i}$$



$$(1 + j0) \cdot (a_{4i} + jb_{4i}) = \textcolor{red}{a_{4i}} + \textcolor{blue}{jb_{4i}}$$

$$\rightarrow \Re = a_{4i}, \quad \Im = b_{4i}$$

$$(0 + j1) \cdot (a_{5i} + jb_{5i}) = \textcolor{red}{-b_{5i}} + \textcolor{blue}{ja_{5i}}$$

$$\rightarrow \Re = -b_{5i}, \quad \Im = a_{5i}$$

$$(-1 + j0) \cdot (a_{6i} + jb_{6i}) = \textcolor{red}{-a_{6i}} - \textcolor{blue}{jb_{6i}}$$

$$\rightarrow \Re = -a_{6i}, \quad \Im = -b_{6i}$$

$$(0 - j1) \cdot (a_{7i} + jb_{7i}) = \textcolor{red}{b_{7i}} - \textcolor{blue}{ja_{7i}}$$

$$\rightarrow \Re = b_{7i}, \quad \Im = -a_{7i}$$

$$\Rightarrow \Re_{2i} = a_{0i} - b_{1i} - a_{2i} + b_{3i} + a_{4i} - b_{5i} - a_{6i} + b_{7i}$$

$$\Rightarrow \Im_{2i} = b_{0i} + a_{1i} - b_{2i} - a_{3i} + b_{4i} + a_{5i} - b_{6i} - a_{7i}$$

#### 4. Zeile:

$$\begin{aligned} & (1 + j0) \cdot (a_{0i} + jb_{0i}) + (-\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}) \cdot (a_{1i} + jb_{1i}) + (0 + j1) \cdot (a_{2i} + jb_{2i}) + (\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}) \cdot (a_{3i} + jb_{3i}) + \\ & (-1 + j0) \cdot (a_{4i} + jb_{4i}) + (\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}) \cdot (a_{5i} + jb_{5i}) + (0 - j1) \cdot (a_{6i} + jb_{6i}) + (-\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}) \cdot (a_{7i} + jb_{7i}) \end{aligned}$$

$$(1 + j0) \cdot (a_{0i} + jb_{0i}) = \textcolor{red}{a_{0i}} + \textcolor{blue}{jb_{0i}}$$

$$\rightarrow \Re = a_{0i}, \quad \Im = b_{0i}$$

$$(-\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}) \cdot (a_{1i} + jb_{1i}) = \textcolor{red}{-\frac{\sqrt{2}}{2} \cdot a_{1i}} + \textcolor{blue}{j\frac{\sqrt{2}}{2} \cdot a_{1i}} - \textcolor{blue}{j\frac{\sqrt{2}}{2} \cdot b_{1i}} - \textcolor{red}{\frac{\sqrt{2}}{2} \cdot b_{1i}}$$

$$\rightarrow \Re = -\frac{\sqrt{2}}{2} \cdot a_{1i} - \frac{\sqrt{2}}{2} \cdot b_{1i}, \quad \Im = \frac{\sqrt{2}}{2} \cdot a_{1i} - \frac{\sqrt{2}}{2} \cdot b_{1i}$$

$$(0 + j1) \cdot (a_{2i} + jb_{2i}) = \textcolor{red}{-b_{2i}} + \textcolor{blue}{a_{2i}}$$

$$\rightarrow \Re = -b_{2i}, \quad \Im = a_{2i}$$

$$(\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}) \cdot (a_{3i} + jb_{3i}) = \textcolor{red}{\frac{\sqrt{2}}{2} \cdot a_{3i}} + \textcolor{blue}{\frac{\sqrt{2}}{2} \cdot a_{3i}} + \textcolor{blue}{\frac{\sqrt{2}}{2} \cdot b_{3i}} - \textcolor{red}{\frac{\sqrt{2}}{2} \cdot b_{3i}}$$

$$\rightarrow \Re = \frac{\sqrt{2}}{2} \cdot a_{3i} - \frac{\sqrt{2}}{2} \cdot b_{3i}, \quad \Im = \frac{\sqrt{2}}{2} \cdot a_{3i} + \frac{\sqrt{2}}{2} \cdot b_{3i}$$

$$\underline{(-1 + j0) \cdot (a_{4i} + jb_{4i})} = \textcolor{red}{-a_{4i}} - \textcolor{blue}{jb_{4i}}$$

$$\rightarrow \Re = -a_{4i}, \quad \Im = -b_{4i}$$

$$\underline{(\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}) \cdot (a_{5i} + jb_{5i})} = \textcolor{red}{\frac{\sqrt{2}}{2} \cdot a_{5i}} - \textcolor{blue}{j\frac{\sqrt{2}}{2} \cdot a_{5i}} + \textcolor{blue}{j\frac{\sqrt{2}}{2} \cdot b_{5i}} + \textcolor{red}{\frac{\sqrt{2}}{2} \cdot b_{5i}}$$

$$\rightarrow \Re = \frac{\sqrt{2}}{2} \cdot a_{5i} + \frac{\sqrt{2}}{2} \cdot b_{5i}, \quad \Im = -\frac{\sqrt{2}}{2} \cdot a_{5i} + \frac{\sqrt{2}}{2} \cdot b_{5i}$$

$$\underline{(0 - j1) \cdot (a_{6i} + jb_{6i})} = \textcolor{red}{b_{6i}} - \textcolor{blue}{ja_{6i}}$$

$$\rightarrow \Re = b_{6i}, \quad \Im = -a_{6i}$$

$$\underline{(-\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}) \cdot (a_{7i} + jb_{7i})} = \textcolor{red}{-\frac{\sqrt{2}}{2} \cdot a_{7i}} - \textcolor{blue}{j\frac{\sqrt{2}}{2} \cdot a_{7i}} - \textcolor{blue}{j\frac{\sqrt{2}}{2} \cdot b_{7i}} + \textcolor{red}{\frac{\sqrt{2}}{2} \cdot b_{7i}}$$

$$\rightarrow \Re = -\frac{\sqrt{2}}{2} \cdot a_{7i} + \frac{\sqrt{2}}{2} \cdot b_{7i}, \quad \Im = -\frac{\sqrt{2}}{2} \cdot a_{7i} - \frac{\sqrt{2}}{2} \cdot b_{7i}$$

$$\Rightarrow \Re_{3i} = a_{0i} - \frac{\sqrt{2}}{2} \cdot a_{1i} - \frac{\sqrt{2}}{2} \cdot b_{1i} - b_{2i} + \frac{\sqrt{2}}{2} \cdot a_{3i} - \frac{\sqrt{2}}{2} \cdot b_{3i} - a_{4i} + \frac{\sqrt{2}}{2} \cdot a_{5i} + \frac{\sqrt{2}}{2} \cdot b_{5i} + b_{6i} - \frac{\sqrt{2}}{2} \cdot a_{7i} + \frac{\sqrt{2}}{2} \cdot b_{7i}$$

$$\Rightarrow \Im_{3i} = b_{0i} + \frac{\sqrt{2}}{2} \cdot a_{1i} - \frac{\sqrt{2}}{2} \cdot b_{1i} + a_{2i} + \frac{\sqrt{2}}{2} \cdot a_{3i} + \frac{\sqrt{2}}{2} \cdot b_{3i} - b_{4i} - \frac{\sqrt{2}}{2} \cdot a_{5i} + \frac{\sqrt{2}}{2} \cdot b_{5i} - a_{6i} - \frac{\sqrt{2}}{2} \cdot a_{7i} - \frac{\sqrt{2}}{2} \cdot b_{7i}$$

5. Zeile:

$$(1 + j0) \cdot (a_{0i} + jb_{0i}) + (-1 + j0) \cdot (a_{1i} + jb_{1i}) + (1 + j0) \cdot (a_{2i} + jb_{2i}) + (-1 + j0) \cdot (a_{3i} + jb_{3i}) + (1 + j0) \cdot (a_{4i} + jb_{4i}) + (-1 + j0) \cdot (a_{5i} + jb_{5i}) + (1 + j0) \cdot (a_{6i} + jb_{6i}) + (-1 + j0) \cdot (a_{7i} + jb_{7i})$$

$$= a_{0i} + jb_{0i} - a_{1i} - jb_{1i} + a_{2i} + jb_{2i} - a_{3i} - jb_{3i} + a_{4i} + jb_{4i} - a_{5i} - jb_{5i} + a_{6i} + jb_{6i} - a_{7i} - +jb_{7i}$$

$$\Rightarrow \Re_{4i} = a_{0i} - a_{1i} + a_{2i} - a_{3i} + a_{4i} - a_{5i} + a_{6i} - a_{7i}$$

$$\Rightarrow \Im_{4i} = b_{0i} - b_{1i} + b_{2i} - b_{3i} + b_{4i} - b_{5i} + b_{6i} - b_{7i}$$

6. Zeile:

$$\underline{(1 + j0) \cdot (a_{0i} + jb_{0i})} + \underline{(-\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}) \cdot (a_{1i} + jb_{1i})} + \underline{(0 + j1) \cdot (a_{2i} + jb_{2i})} + \underline{(\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}) \cdot (a_{3i} + jb_{3i})} + \underline{(-1 + j0) \cdot (a_{4i} + jb_{4i})} + \underline{(\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}) \cdot (a_{5i} + jb_{5i})} + \underline{(0 - j1) \cdot (a_{6i} + jb_{6i})} + \underline{(-\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}) \cdot (a_{7i} + jb_{7i})}$$

$$\underline{(1 + j0) \cdot (a_{0i} + jb_{0i})} = a_{0i} + jb_{0i}$$

$$\rightarrow \Re = a_{0i}, \quad \Im = b_{0i}$$

$$\underline{\left(-\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}\right) \cdot (a_{1i} + jb_{1i})} = -\frac{\sqrt{2}}{2} \cdot a_{1i} - j\frac{\sqrt{2}}{2} \cdot a_{1i} - j\frac{\sqrt{2}}{2} \cdot b_{1i} + \frac{\sqrt{2}}{2} \cdot b_{1i}$$

$$\rightarrow \Re = -\frac{\sqrt{2}}{2} \cdot a_{1i} + \frac{\sqrt{2}}{2} \cdot b_{1i}, \quad \Im = -\frac{\sqrt{2}}{2} \cdot a_{1i} - \frac{\sqrt{2}}{2} \cdot b_{1i}$$

$$\underline{(0 + j1) \cdot (a_{2i} + jb_{2i})} = -b_{2i} + ja_{2i}$$

$$\rightarrow \Re = -b_{2i}, \quad \Im = a_{2i}$$

$$\underline{\left(\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}\right) \cdot (a_{3i} + jb_{3i})} = \frac{\sqrt{2}}{2} \cdot a_{3i} - j\frac{\sqrt{2}}{2} \cdot a_{3i} + j\frac{\sqrt{2}}{2} \cdot b_{3i} + \frac{\sqrt{2}}{2} \cdot b_{3i}$$

$$\rightarrow \Re = \frac{\sqrt{2}}{2} \cdot a_{3i} + \frac{\sqrt{2}}{2} \cdot b_{3i}, \quad \Im = -\frac{\sqrt{2}}{2} \cdot a_{3i} + \frac{\sqrt{2}}{2} \cdot b_{3i}$$

$$\underline{(-1 + j0) \cdot (a_{4i} + jb_{4i})} = -a_{4i} - jb_{4i}$$

$$\rightarrow \Re = -a_{4i}, \quad \Im = -b_{4i}$$

$$\underline{\left(\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}\right) \cdot (a_{5i} + jb_{5i})} = \frac{\sqrt{2}}{2} \cdot a_{5i} + j\frac{\sqrt{2}}{2} \cdot a_{5i} + j\frac{\sqrt{2}}{2} \cdot b_{5i} - \frac{\sqrt{2}}{2} \cdot b_{5i}$$

$$\rightarrow \Re = \frac{\sqrt{2}}{2} \cdot a_{5i} - \frac{\sqrt{2}}{2} \cdot b_{5i}, \quad \Im = \frac{\sqrt{2}}{2} \cdot a_{5i} + \frac{\sqrt{2}}{2} \cdot b_{5i}$$

$$\underline{(0 - j1) \cdot (a_{6i} + jb_{6i})} = b_{6i} - ja_{6i}$$

$$\rightarrow \Re = b_{6i}, \quad \Im = -a_{6i}$$

$$\underline{\left(-\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}\right) \cdot (a_{7i} + jb_{7i})} = -\frac{\sqrt{2}}{2} \cdot a_{7i} + j\frac{\sqrt{2}}{2} \cdot a_{7i} - j\frac{\sqrt{2}}{2} \cdot b_{7i} - \frac{\sqrt{2}}{2} \cdot b_{7i}$$

$$\rightarrow \Re = -\frac{\sqrt{2}}{2} \cdot a_{7i} - \frac{\sqrt{2}}{2} \cdot b_{7i}, \quad \Im = \frac{\sqrt{2}}{2} \cdot a_{7i} - \frac{\sqrt{2}}{2} \cdot b_{7i}$$

$$\Rightarrow \Re_{5i} = a_{0i} - \frac{\sqrt{2}}{2} \cdot a_{1i} + \frac{\sqrt{2}}{2} \cdot b_{1i} - b_{2i} + \frac{\sqrt{2}}{2} \cdot a_{3i} + \frac{\sqrt{2}}{2} \cdot b_{3i} - a_{4i} + \frac{\sqrt{2}}{2} \cdot a_{5i} - \frac{\sqrt{2}}{2} \cdot b_{5i} + b_{6i} - \frac{\sqrt{2}}{2} \cdot a_{7i} - \frac{\sqrt{2}}{2} \cdot b_{7i}$$

$$\Rightarrow \Im_{5i} = b_{0i} - \frac{\sqrt{2}}{2} \cdot a_{1i} - \frac{\sqrt{2}}{2} \cdot b_{1i} + a_{2i} - \frac{\sqrt{2}}{2} \cdot a_{3i} + \frac{\sqrt{2}}{2} \cdot b_{3i} - b_{4i} + \frac{\sqrt{2}}{2} \cdot a_{5i} + \frac{\sqrt{2}}{2} \cdot b_{5i} - a_{6i} + \frac{\sqrt{2}}{2} \cdot a_{7i} - \frac{\sqrt{2}}{2} \cdot b_{7i}$$

7. Zeile:

$$\underline{(1 + j0) \cdot (a_{0i} + jb_{0i})} + \underline{(0 - j1) \cdot (a_{1i} + jb_{1i})} + \underline{(-1 + j0) \cdot (a_{2i} + jb_{2i})} + \underline{(0 + j1) \cdot (a_{3i} + jb_{3i})} + \underline{(1 + j0) \cdot (a_{4i} + jb_{4i})} + \underline{(0 - j1) \cdot (a_{5i} + jb_{5i})} + \underline{(-1 + j0) \cdot (a_{6i} + jb_{6i})} + \underline{(0 + j1) \cdot (a_{7i} + jb_{7i})}$$

$$\underline{(1 + j0) \cdot (a_{0i} + jb_{0i})} = a_{0i} + jb_{0i}$$

$$\rightarrow \Re = a_{0i}, \quad \Im = b_{0i}$$

$$\underline{(0 - j1) \cdot (a_{1i} + jb_{1i})} = \textcolor{red}{b_{1i}} - \textcolor{blue}{ja_{1i}}$$

$$\rightarrow \Re = b_{1i}, \quad \Im = -a_{1i}$$

$$\underline{(-1 + j0) \cdot (a_{2i} + jb_{2i})} = -\textcolor{red}{a_{2i}} - \textcolor{blue}{jb_{2i}}$$

$$\rightarrow \Re = -a_{2i}, \quad \Im = -b_{2i}$$

$$\underline{(0 + j1) \cdot (a_{3i} + jb_{3i})} = -\textcolor{red}{b_{3i}} + \textcolor{blue}{ja_{3i}}$$

$$\rightarrow \Re = -b_{3i}, \quad \Im = a_{3i}$$

$$\underline{(1 + j0) \cdot (a_{4i} + jb_{4i})} = \textcolor{red}{a_{4i}} + \textcolor{blue}{jb_{4i}}$$

$$\rightarrow \Re = a_{4i}, \quad \Im = b_{4i}$$

$$\underline{(0 - j1) \cdot (a_{5i} + jb_{5i})} = \textcolor{red}{b_{5i}} - \textcolor{blue}{ja_{5i}}$$

$$\rightarrow \Re = b_{5i}, \quad \Im = -a_{5i}$$

$$\underline{(-1 + j0) \cdot (a_{6i} + jb_{6i})} = -\textcolor{red}{a_{6i}} - \textcolor{blue}{jb_{6i}}$$

$$\rightarrow \Re = -a_{6i}, \quad \Im = -b_{6i}$$

$$\underline{(0 + j1) \cdot (a_{7i} + jb_{7i})} = -\textcolor{red}{b_{7i}} + \textcolor{blue}{a_{7i}}$$

$$\rightarrow \Re = -b_{7i}, \quad \Im = a_{7i}$$

$$\Rightarrow \Re_{6i} = a_{0i} + b_{1i} - a_{2i} - b_{3i} + a_{4i} + b_{5i} - a_{6i} - b_{7i}$$

$$\Rightarrow \Im_{6i} = b_{0i} - a_{1i} - b_{2i} + a_{3i} + b_{4i} - a_{5i} - b_{6i} + a_{7i}$$

## 8. Zeile

$$\underline{(1 + j0) \cdot (a_{0i} + jb_{0i})} + \underline{(\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}) \cdot (a_{1i} + jb_{1i})} + \underline{(0 - j1) \cdot (a_{2i} + jb_{2i})} + \underline{(-\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}) \cdot (a_{3i} + jb_{3i})} + \underline{(-1 + j0) \cdot (a_{4i} + jb_{4i})} + \underline{(-\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}) \cdot (a_{5i} + jb_{5i})} + \underline{(0 + j1) \cdot (a_{6i} + jb_{6i})} + \underline{(\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}) \cdot (a_{7i} + jb_{7i})}$$

$$\underline{(1 + j0) \cdot (a_{0i} + jb_{0i})} = \textcolor{red}{a_{0i}} + \textcolor{blue}{jb_{0i}}$$

$$\rightarrow \Re = a_{0i}, \quad \Im = b_{0i}$$

$$\left(\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}\right) \cdot (a_{1i} + jb_{1i}) = \frac{\sqrt{2}}{2} \cdot a_{1i} - j\frac{\sqrt{2}}{2} \cdot a_{1i} + j\frac{\sqrt{2}}{2} \cdot b_{1i} + b_{1i}$$

$$\rightarrow \Re = \frac{\sqrt{2}}{2} \cdot a_{1i} + \frac{\sqrt{2}}{2} \cdot b_{1i}, \quad \Im = -\frac{\sqrt{2}}{2} \cdot a_{1i} + \frac{\sqrt{2}}{2} \cdot b_{1i}$$

$$(0 - j1) \cdot (a_{2i} + jb_{2i}) = b_{2i} - ja_{2i}$$

$$\rightarrow \Re = b_{2i}, \quad \Im = -a_{2i}$$

$$\left(-\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}\right) \cdot (a_{3i} + jb_{3i}) = -\frac{\sqrt{2}}{2} \cdot a_{3i} - j\frac{\sqrt{2}}{2} \cdot a_{3i} - j\frac{\sqrt{2}}{2} \cdot b_{3i} + \frac{\sqrt{2}}{2} \cdot b_{3i}$$

$$\rightarrow \Re = -\frac{\sqrt{2}}{2} \cdot a_{3i} + \frac{\sqrt{2}}{2} \cdot b_{3i}, \quad \Im = -\frac{\sqrt{2}}{2} \cdot a_{3i} - \frac{\sqrt{2}}{2} \cdot b_{3i}$$

$$(-1 + j0) \cdot (a_{4i} + jb_{4i}) = -a_{4i} - jb_{4i}$$

$$\rightarrow \Re = -a_{4i}, \quad \Im = -b_{4i}$$

$$\left(-\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}\right) \cdot (a_{5i} + jb_{5i}) = -\frac{\sqrt{2}}{2} \cdot a_{5i} + j\frac{\sqrt{2}}{2} \cdot a_{5i} - j\frac{\sqrt{2}}{2} \cdot b_{5i} - \frac{\sqrt{2}}{2} \cdot b_{5i}$$

$$\rightarrow \Re = -\frac{\sqrt{2}}{2} \cdot a_{5i} - \frac{\sqrt{2}}{2} \cdot b_{5i}, \quad \Im = \frac{\sqrt{2}}{2} \cdot a_{5i} - \frac{\sqrt{2}}{2} \cdot b_{5i}$$

$$(0 + j1) \cdot (a_{6i} + jb_{6i}) = -b_{6i} + ja_{6i}$$

$$\rightarrow \Re = -b_{6i}, \quad \Im = a_{6i}$$

$$\left(\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}\right) \cdot (a_{7i} + jb_{7i}) = \frac{\sqrt{2}}{2} \cdot a_{7i} + j\frac{\sqrt{2}}{2} \cdot a_{7i} + j\frac{\sqrt{2}}{2} \cdot b_{7i} - \frac{\sqrt{2}}{2} \cdot b_{7i}$$

$$\rightarrow \Re = \frac{\sqrt{2}}{2} \cdot a_{7i} - \frac{\sqrt{2}}{2} \cdot b_{7i}, \quad \Im = \frac{\sqrt{2}}{2} \cdot a_{7i} + \frac{\sqrt{2}}{2} \cdot b_{7i}$$

$$\Rightarrow \Re_{7i} = a_{0i} + \frac{\sqrt{2}}{2} \cdot a_{1i} + \frac{\sqrt{2}}{2} \cdot b_{1i} + b_{2i} - \frac{\sqrt{2}}{2} \cdot a_{3i} + \frac{\sqrt{2}}{2} \cdot b_{3i} - a_{4i} - \frac{\sqrt{2}}{2} \cdot a_{5i} - \frac{\sqrt{2}}{2} \cdot b_{5i} - b_{6i} + \frac{\sqrt{2}}{2} \cdot a_{7i} - \frac{\sqrt{2}}{2} \cdot b_{7i}$$

$$\Rightarrow \Im_{7i} = b_{0i} - \frac{\sqrt{2}}{2} \cdot a_{1i} + \frac{\sqrt{2}}{2} \cdot b_{1i} - a_{2i} - \frac{\sqrt{2}}{2} \cdot a_{3i} - \frac{\sqrt{2}}{2} \cdot b_{3i} - b_{4i} + \frac{\sqrt{2}}{2} \cdot a_{5i} - \frac{\sqrt{2}}{2} \cdot b_{5i} + a_{6i} + \frac{\sqrt{2}}{2} \cdot a_{7i} + \frac{\sqrt{2}}{2} \cdot b_{7i}$$

Umsortieren ergibt:

$$\Re_{0i} = \underbrace{a_{0i} + a_{1i}}_{\text{sum0\_stage1\_1v4\_re}} + \underbrace{a_{2i} + a_{3i}}_{\text{sum0\_stage1\_2v4\_re}} + \underbrace{a_{4i} + a_{5i}}_{\text{sum0\_stage1\_3v4\_re}} + \underbrace{a_{6i} + a_{7i}}_{\text{sum0\_stage1\_4v4\_re}}$$

$$\Im_{0i} = \underbrace{b_{0i} + b_{1i}}_{\text{sum0\_stage1\_1v4\_im}} + \underbrace{b_{2i} + b_{3i}}_{\text{sum0\_stage1\_2v4\_im}} + \underbrace{b_{4i} + b_{5i}}_{\text{sum0\_stage1\_3v4\_im}} + \underbrace{b_{6i} + b_{7i}}_{\text{sum0\_stage1\_4v4\_im}}$$

$$\begin{aligned}\Re_{1i} = & \underbrace{a_{0i} - \frac{\sqrt{2}}{2} \cdot b_{1i}}_{\text{sum1\_stage1\_1v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{1i} - b_{2i}}_{\text{sum1\_stage1\_2v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{5i} - \frac{\sqrt{2}}{2} \cdot a_{3i}}_{\text{sum1\_stage1\_3v6\_re}} \\ & + \underbrace{b_{6i} - \frac{\sqrt{2}}{2} \cdot b_{3i}}_{\text{sum1\_stage1\_4v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{7i} - a_{4i}}_{\text{sum1\_stage1\_5v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{7i} - \frac{\sqrt{2}}{2} \cdot a_{5i}}_{\text{sum1\_stage1\_6v6\_re}}\end{aligned}$$

$$\begin{aligned}\Im_{1i} = & \underbrace{b_{0i} - \frac{\sqrt{2}}{2} \cdot b_{3i}}_{\text{sum1\_stage1\_1v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{1i} - b_{4i}}_{\text{sum1\_stage1\_2v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{1i} - \frac{\sqrt{2}}{2} \cdot a_{5i}}_{\text{sum1\_stage1\_3v6\_im}} \\ & + \underbrace{a_{2i} - \frac{\sqrt{2}}{2} \cdot b_{5i}}_{\text{sum1\_stage1\_4v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{3i} - a_{6i}}_{\text{sum1\_stage1\_5v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{7i} - \frac{\sqrt{2}}{2} \cdot a_{7i}}_{\text{sum1\_stage1\_5v6\_im}}\end{aligned}$$

$$\Re_{2i} = \underbrace{a_{0i} - b_{1i}}_{\text{sum2\_stage1\_1v4\_re}} + \underbrace{b_{3i} - a_{2i}}_{\text{sum2\_stage1\_2v4\_re}} + \underbrace{a_{4i} - b_{5i}}_{\text{sum2\_stage1\_3v4\_re}} + \underbrace{b_{7i} - a_{6i}}_{\text{sum2\_stage1\_4v4\_re}}$$

$$\Im_{2i} = \underbrace{b_{0i} - b_{2i}}_{\text{sum2\_stage1\_1v4\_im}} + \underbrace{a_{1i} - a_{3i}}_{\text{sum2\_stage1\_2v4\_im}} + \underbrace{b_{4i} - b_{6i}}_{\text{sum2\_stage1\_3v4\_im}} + \underbrace{a_{5i} - a_{7i}}_{\text{sum2\_stage1\_4v4\_im}}$$

$$\begin{aligned}\Re_{3i} = & \underbrace{a_{0i} - \frac{\sqrt{2}}{2} \cdot a_{1i}}_{\text{sum3\_stage1\_1v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{3i} - \frac{\sqrt{2}}{2} \cdot b_{1i}}_{\text{sum3\_stage1\_2v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{5i} - b_{2i}}_{\text{sum3\_stage1\_3v6\_re}} \\ & + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{5i} - \frac{\sqrt{2}}{2} \cdot b_{3i}}_{\text{sum3\_stage1\_4v6\_re}} + \underbrace{b_{6i} - a_{4i}}_{\text{sum3\_stage1\_5v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{7i} - \frac{\sqrt{2}}{2} \cdot a_{7i}}_{\text{sum3\_stage1\_6v6\_re}}\end{aligned}$$

$$\begin{aligned}\Im_{3i} = & \underbrace{b_{0i} - \frac{\sqrt{2}}{2} \cdot b_{1i}}_{\text{sum3\_stage1\_1v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{1i} - b_{4i}}_{\text{sum3\_stage1\_2v6\_im}} + \underbrace{a_{2i} - \frac{\sqrt{2}}{2} \cdot a_{5i}}_{\text{sum3\_stage1\_3v6\_im}} \\ & + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{3i} - a_{6i}}_{\text{sum3\_stage1\_4v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{3i} - \frac{\sqrt{2}}{2} \cdot a_{7i}}_{\text{sum3\_stage1\_5v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{5i} - \frac{\sqrt{2}}{2} \cdot b_{7i}}_{\text{sum3\_stage1\_6v6\_im}}\end{aligned}$$

$$\Re_{4i} = \underbrace{a_{0i} - a_{1i}}_{\text{sum4\_stage1\_1v4\_re}} + \underbrace{a_{2i} - a_{3i}}_{\text{sum4\_stage1\_2v4\_re}} + \underbrace{a_{4i} - a_{5i}}_{\text{sum4\_stage1\_3v4\_re}} + \underbrace{a_{6i} - a_{7i}}_{\text{sum4\_stage1\_4v4\_re}}$$

$$\Im_{4i} = \underbrace{b_{0i} - b_{1i}}_{\text{sum4\_stage1\_1v4\_im}} + \underbrace{b_{2i} - b_{3i}}_{\text{sum4\_stage1\_2v4\_im}} + \underbrace{b_{4i} - b_{5i}}_{\text{sum4\_stage1\_3v4\_im}} + \underbrace{b_{6i} - b_{7i}}_{\text{sum4\_stage1\_4v4\_im}}$$

$$\begin{aligned} \Re_{5i} = & \underbrace{a_{0i} - \frac{\sqrt{2}}{2} \cdot a_{1i}}_{\text{sum5\_stage1\_1v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{1i} - b_{2i}}_{\text{sum5\_stage1\_2v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{3i} - a_{4i}}_{\text{sum5\_stage1\_3v6\_re}} \\ & + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{3i} - \frac{\sqrt{2}}{2} \cdot b_{5i}}_{\text{sum5\_stage1\_4v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{5i} - \frac{\sqrt{2}}{2} \cdot a_{7i}}_{\text{sum5\_stage1\_5v6\_re}} + \underbrace{b_{6i} - \frac{\sqrt{2}}{2} \cdot b_{7i}}_{\text{sum5\_stage1\_6v6\_re}} \end{aligned}$$

$$\begin{aligned} \Im_{5i} = & \underbrace{b_{0i} - \frac{\sqrt{2}}{2} \cdot a_{1i}}_{\text{sum5\_stage1\_1v6\_im}} + \underbrace{a_{2i} - \frac{\sqrt{2}}{2} \cdot b_{1i}}_{\text{sum5\_stage1\_2v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{3i} - \frac{\sqrt{2}}{2} \cdot a_{3i}}_{\text{sum5\_stage1\_3v6\_im}} \\ & + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{5i} - b_{4i}}_{\text{sum5\_stage1\_4v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{5i} - a_{6i}}_{\text{sum5\_stage1\_5v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{7i} - \frac{\sqrt{2}}{2} \cdot b_{7i}}_{\text{sum5\_stage1\_6v6\_im}} \end{aligned}$$

$$\Re_{6i} = \underbrace{a_{0i} - a_{2i}}_{\text{sum6\_stage1\_1v4\_re}} + \underbrace{b_{1i} - b_{3i}}_{\text{sum6\_stage1\_2v4\_re}} + \underbrace{a_{4i} - a_{6i}}_{\text{sum6\_stage1\_3v4\_re}} + \underbrace{b_{5i} - b_{7i}}_{\text{sum6\_stage1\_4v4\_re}}$$

$$\Im_{6i} = \underbrace{b_{0i} - a_{1i}}_{\text{sum6\_stage1\_1v4\_im}} + \underbrace{a_{3i} - b_{2i}}_{\text{sum6\_stage1\_2v4\_im}} + \underbrace{b_{4i} - a_{5i}}_{\text{sum6\_stage1\_3v4\_im}} + \underbrace{a_{7i} - b_{6i}}_{\text{sum6\_stage1\_4v4\_im}}$$

$$\begin{aligned} \Re_{7i} = & \underbrace{a_{0i} - \frac{\sqrt{2}}{2} \cdot a_{3i}}_{\text{sum7\_stage1\_1v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{1i} - a_{4i}}_{\text{sum7\_stage1\_2v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{1i} - \frac{\sqrt{2}}{2} \cdot a_{5i}}_{\text{sum7\_stage1\_3v6\_re}} \\ & + \underbrace{b_{2i} - \frac{\sqrt{2}}{2} \cdot b_{5i}}_{\text{sum7\_stage1\_4v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{3i} - b_{6i}}_{\text{sum7\_stage1\_5v6\_re}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{7i} - \frac{\sqrt{2}}{2} \cdot b_{7i}}_{\text{sum7\_stage1\_6v6\_re}} \end{aligned}$$

$$\begin{aligned} \Im_{7i} = & \underbrace{b_{0i} - \frac{\sqrt{2}}{2} \cdot a_{1i}}_{\text{sum7\_stage1\_1v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{1i} - a_{2i}}_{\text{sum7\_stage1\_2v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{5i} - \frac{\sqrt{2}}{2} \cdot a_{3i}}_{\text{sum7\_stage1\_3v6\_im}} \\ & + \underbrace{a_{6i} - \frac{\sqrt{2}}{2} \cdot b_{3i}}_{\text{sum7\_stage1\_4v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot a_{7i} - b_{4i}}_{\text{sum7\_stage1\_5v6\_im}} + \underbrace{\frac{\sqrt{2}}{2} \cdot b_{7i} - \frac{\sqrt{2}}{2} \cdot b_{5i}}_{\text{sum7\_stage1\_6v6\_im}} \end{aligned}$$

## 8.6 Programmcode

```

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 8.9: Deklaration der Konstanten

```

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

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 8.10: Deklaration eigener Datentypen

```

library IEEE;
2  use ieee.std_logic_1164.all;
   --use ieee.std_logic_arith.all;
4  use ieee.numeric_std.all;

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

10 library work;
   use work.all;
12 use datatypes.all;
   use constants.all;
14

16 entity read_input_matrix is
    port(
18         clk           : in  bit;
           loaded        : out bit;
20         input_real    : out t_2d_array;
           input_imag    : out t_2d_array
22     );
end entity read_input_matrix;
24

26 architecture bhv of read_input_matrix is
begin
28     reading : process

30         variable element_1_real : std_logic_vector(bit_width_extern-1 downto 0) := (
           others => '0');
           variable element_1_imag : std_logic_vector(bit_width_extern-1 downto 0) := (
           others => '0');
32         variable element_2_real : std_logic_vector(bit_width_extern-1 downto 0) := (
           others => '0');
           variable element_2_imag : std_logic_vector(bit_width_extern-1 downto 0) := (
           others => '0');
34         variable element_3_real : std_logic_vector(bit_width_extern-1 downto 0) := (
           others => '0');
           variable element_3_imag : std_logic_vector(bit_width_extern-1 downto 0) := (
           others => '0');
36         variable element_4_real : std_logic_vector(bit_width_extern-1 downto 0) := (
           others => '0');
           variable element_4_imag : std_logic_vector(bit_width_extern-1 downto 0) := (
           others => '0');
38         variable element_5_real : std_logic_vector(bit_width_extern-1 downto 0) := (

```

```

others => '0');
variable element_5_imag : std_logic_vector(bit_width_extern-1 downto 0) := (
others => '0');
40 variable element_6_real : std_logic_vector(bit_width_extern-1 downto 0) := (
others => '0');
variable element_6_imag : std_logic_vector(bit_width_extern-1 downto 0) := (
others => '0');
42 variable element_7_real : std_logic_vector(bit_width_extern-1 downto 0) := (
others => '0');
variable element_7_imag : std_logic_vector(bit_width_extern-1 downto 0) := (
others => '0');
44 variable element_8_real : std_logic_vector(bit_width_extern-1 downto 0) := (
others => '0');
variable element_8_imag : std_logic_vector(bit_width_extern-1 downto 0) := (
others => '0');
46
variable r_space : character;
48
variable fstatus : file_open_status; — status r,w
variable inline : line; — readout line
50 file infile : text; — filehandle for reading ascii text
52
variable textfilename : string(1 to 29);
54
begin
56
58
if bit_width_extern = 12 then
60 textfilename := "InputMatrix_komplex_12Bit.txt";
else
62 textfilename := "InputMatrix_komplex_16Bit.txt";
end if;
64
file_open(fstatus,infile , textfilename , read_mode);
66
— if fstatus = NAME_ERROR then
68 — file_open(fstatus,infile , "HDL/InputMatrix_komplex.txt", read_mode);
— —report "Ausgabe-Datei befindet sich im Unterverzeichnis 'HDL'.";
70 — end if;
72
for i in 0 to mat_size-1 loop
74
wait until clk = '1' and clk'event;
readline(infile , inline);
76 read(inline , element_1_real);
read(inline , r_space);
78 read(inline , element_1_imag);
read(inline , r_space);
80 read(inline , element_2_real);
read(inline , r_space);
82 read(inline , element_2_imag);
read(inline , r_space);
84 read(inline , element_3_real);
read(inline , r_space);
86 read(inline , element_3_imag);
read(inline , r_space);
88 read(inline , element_4_real);

```

```

90      read (inline , r_space);
      read (inline , element_4_imag);
      read (inline , r_space);
92      read (inline , element_5_real);
      read (inline , r_space);
94      read (inline , element_5_imag);
      read (inline , r_space);
96      read (inline , element_6_real);
      read (inline , r_space);
98      read (inline , element_6_imag);
      read (inline , r_space);
100     read (inline , element_7_real);
      read (inline , r_space);
102     read (inline , element_7_imag);
      read (inline , r_space);
104     read (inline , element_8_real);
      read (inline , r_space);
106     read (inline , element_8_imag);

108     input_real(i)(0) <= signed(element_1_real);
      input_imag(i)(0) <= signed(element_1_imag);
110     input_real(i)(1) <= signed(element_2_real);
      input_imag(i)(1) <= signed(element_2_imag);
112     input_real(i)(2) <= signed(element_3_real);
      input_imag(i)(2) <= signed(element_3_imag);
114     input_real(i)(3) <= signed(element_4_real);
      input_imag(i)(3) <= signed(element_4_imag);
116     input_real(i)(4) <= signed(element_5_real);
      input_imag(i)(4) <= signed(element_5_imag);
118     input_real(i)(5) <= signed(element_6_real);
      input_imag(i)(5) <= signed(element_6_imag);
120     input_real(i)(6) <= signed(element_7_real);
      input_imag(i)(6) <= signed(element_7_imag);
122     input_real(i)(7) <= signed(element_8_real);
      input_imag(i)(7) <= signed(element_8_imag);

124     if i = mat_size-1 then
126         loaded <= '1' after 10 ns;
      end if;
128 end loop;
      file_close(infile);
130 wait;

132 end process;
134 end bhv;

```

Listing 8.11: Eingangs-Matrix aus Textdatei einlesen

```

library ieee;
2 use ieee.std_logic_1164.all;
  use ieee.std_logic_arith.all;
4 library work;
  use work.all;
6 use datatypes.all;

8 entity read_input_matrix_tb is
end entity read_input_matrix_tb;

```

```

10 architecture arch of read_input_matrix_tb is
12
13     signal clk          : bit := '0';
14     signal loaded       : bit := '0';
15     signal input_real   : t_2d_array;
16     signal input_imag   : t_2d_array;
17
18     component read_input_matrix is
19         port(
20             clk          : in  bit;
21             loaded       : out bit;
22             input_real   : out t_2d_array;
23             input_imag   : out t_2d_array;
24         );
25     end component;
26
27     begin
28         dut : read_input_matrix
29             port map(
30                 clk          => clk ,
31                 loaded       => loaded ,
32                 input_real   => input_real ,
33                 input_imag   => input_imag
34             );
35
36         clk <= not clk after 20 ns;
37     end arch;

```

Listing 8.12: Testbench für das Einlesen aus einer Textdatei

```

library IEEE;
2 use ieee.std_logic_1164.all;
  —use ieee.std_logic_arith.all;
4 use ieee.numeric_std.all;

6 library STD; — for writing text file
  use STD.TEXTIO.ALL;
8 use ieee.std_logic_textio.all;

10 library work;
  use work.all;
12 use datatypes.all;
  use constants.all;
14

16
17
18 entity write_results is
19     port(
20         result_ready : in  bit;
21         result_real   : in  t_2d_array;
22         result_imag   : in  t_2d_array;
23         write_done    : out bit;
24     );
25 end entity write_results;

26
27 architecture bhv of write_results is

```

```

28 begin
    writing_to_file : process(result_ready)
30
    variable fstatus : file_open_status; — status r,w
32    variable outline : line; — writeout line
    file      outfile : text; — filehandle
34
    —variable output1 : bit_vector(3 downto 0) := "0101";
36    —variable output2 : bit_vector(3 downto 0) := "0110";
38
    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);
40    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);
42    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);
44    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);
46    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);
48    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);
50    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);
52    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);
54    variable space : character := ' ';
56
    begin
58        file_open(fstatus, outfile, "/home/tlattmann/cadence/mat_mult/HDL/Results.txt"
        , write_mode);
60
        —if result_ready = '1' then
62        for i in 0 to mat_size-1 loop
            element_1_real := std_logic_vector(result_real(i)(0));
            element_1_imag := std_logic_vector(result_imag(i)(0));
            element_2_real := std_logic_vector(result_real(i)(1));
            element_2_imag := std_logic_vector(result_imag(i)(1));
            element_3_real := std_logic_vector(result_real(i)(2));
            element_3_imag := std_logic_vector(result_imag(i)(2));
            element_4_real := std_logic_vector(result_real(i)(3));
            element_4_imag := std_logic_vector(result_imag(i)(3));
            element_5_real := std_logic_vector(result_real(i)(4));
            element_5_imag := std_logic_vector(result_imag(i)(4));
            element_6_real := std_logic_vector(result_real(i)(5));
            element_6_imag := std_logic_vector(result_imag(i)(5));
            element_7_real := std_logic_vector(result_real(i)(6));
            element_7_imag := std_logic_vector(result_imag(i)(6));
            element_8_real := std_logic_vector(result_real(i)(7));
            element_8_imag := std_logic_vector(result_imag(i)(7));
80
            write(outline, element_1_real);
            write(outline, space);
            write(outline, element_1_imag);
            write(outline, space);
82
            write(outline, element_2_real);
84

```

```

      write(outline, space);
86      write(outline, element_2_imag);
      write(outline, space);
88      write(outline, element_3_real);
      write(outline, space);
90      write(outline, element_3_imag);
      write(outline, space);
92      write(outline, element_4_real);
      write(outline, space);
94      write(outline, element_4_imag);
      write(outline, space);
96      write(outline, element_5_real);
      write(outline, space);
98      write(outline, element_5_imag);
      write(outline, space);
100     write(outline, element_6_real);
      write(outline, space);
102     write(outline, element_6_imag);
      write(outline, space);
104     write(outline, element_7_real);
      write(outline, space);
106     write(outline, element_7_imag);
      write(outline, space);
108     write(outline, element_8_real);
      write(outline, space);
110     write(outline, element_8_imag);

112     writeline(outfile, outline);
end loop;
114

write_done <= '1';
file_close(outfile);
—end if;
118

end process;
120 end bhv;

```

Listing 8.13: Ergebnis-Matrix in Textdatei schreiben

```

library IEEE;
2 use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
4
library STD; — for writing 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 write_test_tb is
16 end entity write_test_tb;

18
architecture bhv of write_test_tb is

```

```

20  signal clk          : bit;
22  signal loaded       : bit;
24  signal result_ready : bit;
26  signal write_done   : bit;
28  signal loop_running : bit;
30  signal loop_number  : signed(2 downto 0);
32  signal input_real   : t_2d_array;
34  signal input_imag   : t_2d_array;
36  signal output       : std_logic_vector(bit_width_extern-1 downto 0);
38
39  component read_input_matrix
40  port(
41      clk          : in  bit;
42      loaded       : out bit;
43      input_real   : out t_2d_array;
44      input_imag   : out t_2d_array
45  );
46  end component;
47
48  component write_results
49  port(
50      result_ready : in  bit;
51      result_real  : in  t_2d_array;
52      result_imag  : in  t_2d_array;
53      write_done   : out bit;
54      loop_number  : out signed(2 downto 0);
55      loop_running : out bit;
56      output       : out std_logic_vector(bit_width_extern-1 downto 0)
57  );
58  end component;
59
60  begin
61
62  mat : read_input_matrix
63  port map(
64      clk          => clk ,
65      loaded       => loaded ,
66      input_real   => input_real ,
67      input_imag   => input_imag
68  );
69
70  write : write_results
71  port map(
72      result_ready => result_ready ,
73      result_real  => input_real ,
74      result_imag  => input_imag ,
75      write_done   => write_done ,
76      loop_number  => loop_number ,
77      loop_running => loop_running ,
78      output       => output
79  );
80
81  result_ready <= loaded after 20 ns;
82  clk          <= not clk after 10 ns;
83
84  end bhv;

```

Listing 8.14: Testbench für das schreiben in eine Textdatei

```

library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
  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
38     begin
39         FSM_TAKT: process(clk)
40         begin
41             if clk='1' and clk'event then
42                 dft_state <= dft_state;
43                 state_out <= dft_state;
44                 if nReset='0' then
45                     dft_state <= idle;
46                     state_out <= idle;
47                 elsif loaded = '0' then
48                     dft_state <= idle;
49                     state_out <= idle;
50                 elsif loaded='1' and dft_state = idle then
51                     dft_state <= twiddle_calc;
52                     state_out <= twiddle_calc;
53                 else
54                     dft_state <= next_dft_state;
55                     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(bit_width_adder-1 downto 0) :=
63     "0001011010100";
64     variable twiddle_coeff : signed(16 downto 0) := "00010110101000001";
65
66     variable mult_re, mult_im : signed(bit_width_multiplier-1 downto 0);
67
68     variable W_row, I_col : integer;
69     variable dft_1d_real, dft_1d_imag : t_2d_array;
70     variable matrix_real, matrix_imag : t_2d_array;
71     variable temp_re, temp_im : t_1d_array6_13bit;
72     variable temp14bit_re, temp14bit_im : signed(bit_width_adder downto 0);
73     variable dft_1d_2d : bit;
74     variable element : unsigned(5 downto 0) := "000000";
75
76     variable row_col_idx : integer := 0;
77
78     variable LineBuffer : LINE;
79
80 begin
81     — Flip-Flops
82     — werden das 1. Mal sich selbst zu gewiesen, bevor sie einen Wert haben!
83     result_ready <= '0';
84     element := element;
85     dft_1d_2d := dft_1d_2d;
86     temp_re := temp_re;
87     temp_im := temp_im;
88     mult_re := mult_re;
89     mult_im := mult_im;
90     dft_1d_real := dft_1d_real;
91     dft_1d_imag := dft_1d_imag;
92     matrix_real := matrix_real;
93     matrix_imag := matrix_imag;
94     dft_1d_2d_out <= dft_1d_2d;
95
96
97     — Die Matrix hat 64 Elemente -> 2^6=64 -> 6-Bit Vektor passt genau. Ueberlauf =
98     1. Element vom n chsten Durchlauf.
99     — Der Elemente-Vektor kann darueber hinaus in vordere Haelfte = Zeile und
100     hintere Haelfte = Spalte aufgeteilt werden.
101     — So laesst sich auch ein Matrix-Element mit zwei Indizes ansprechen:
102
103     — Bei der IDFT sind die Zeilen 1 und 7, 2 und 6, 3 und 5 vertauscht. 1 und 4
104     bleiben wie sie sind.
105
106     row_col_idx := to_integer(element(5 downto 3)); — Wird bei der Twiddlefaktor-
107     Matrix als Zeilen-, bei der Zwischen- und — Ausgangsmatrix als
108     Spaltenindex verwendet.
109
110     if idft = '1' then

```

```

108     if row_col_idx = 0 then
109         W_row := 0;
110     else
111         W_row := 8-row_col_idx; — Twiddlefaktor-Matrix
112     end if;
113 else
114     W_row := row_col_idx; — Twiddlefaktor-Matrix
115 end if;
116
117 I_col := to_integer(element(2 downto 0)); — Input-Matrix
118
119 if element = "000000" then
120     if dft_ld_2d = '0' then
121         matrix_real := input_real;
122         matrix_imag := input_imag;
123     else
124         matrix_real := dft_ld_real;
125         matrix_imag := dft_ld_imag;
126     end if;
127 end if;
128
129 case dft_state is
130     when idle =>
131         next_dft_state <= twiddle_calc;
132
133     when twiddle_calc => — dft_state_out = 1
134         — Mit resize werden die 12 Bit Eingangswerte vorzeichengerecht auf 13 Bit
135         — erweitert, um die richtige Groesse zu haben.
136         — Bei der Addition muessen die Summanden die gleiche Bit-Breite wie der
137         — Ergebnis-Vektor haben.
138         case W_row is
139             — Die Faktoren (Koeffizienten) der Twiddlefaktor-Matrix W lassen sich
140             — ueber  $\exp(i \cdot 2 \cdot \pi \cdot [0:7]' \cdot [0:7]/8)$  berechnen.
141             — 1. Zeile aus W -> nur Additionen
142             when 0 =>
143                 — Die 1. Zeile aus W besteht nur aus den Faktoren  $(1+j0)$ . Daraus
144                 — resultiert, dass die reellen
145                 — und die imaginären Werte der Eingangs-Matrix unabhaengig von
146                 — einander aufsummiert werden.
147                 — Real
148                 temp_re(0) := resize(matrix_real(0)(I_col), bit_width_adder) + resize(
149                 matrix_real(1)(I_col), bit_width_adder);
150                 temp_re(1) := resize(matrix_real(2)(I_col), bit_width_adder) + resize(
151                 matrix_real(3)(I_col), bit_width_adder);
152                 temp_re(2) := resize(matrix_real(4)(I_col), bit_width_adder) + resize(
153                 matrix_real(5)(I_col), bit_width_adder);
154                 temp_re(3) := resize(matrix_real(6)(I_col), bit_width_adder) + resize(
155                 matrix_real(7)(I_col), bit_width_adder);
156                 — Imag
157                 temp_im(0) := resize(matrix_imag(0)(I_col), bit_width_adder) + resize(
158                 matrix_imag(1)(I_col), bit_width_adder);
159                 temp_im(1) := resize(matrix_imag(2)(I_col), bit_width_adder) + resize(
160                 matrix_imag(3)(I_col), bit_width_adder);
161                 temp_im(2) := resize(matrix_imag(4)(I_col), bit_width_adder) + resize(
162                 matrix_imag(5)(I_col), bit_width_adder);
163                 temp_im(3) := resize(matrix_imag(6)(I_col), bit_width_adder) + resize(

```

```

matrix_imag(7)(I_col), bit_width_adder);

-- 2. Zeile aus W besteht aus den Faktoren
-- 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)

-- Wegen der Faktoren (+/-0.70711 +/-0.70711i) haben die geraden Zeilen (
beginnend bei 1) 12 statt 8 Subtraktionen
-- 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
-- insbesondere auch den Platzbedarf auf dem Chip klein zuhalten, wird die
Multiplikation auf die Summe aller und
-- nicht auf die einzelnen Werte angewandt.
-- Da immer genau die Haelfte der Faktoren positiv und die andere negativ
ist, werden die Eingangswerte so sortiert,
-- dass keine Negationen noetig sind.
when 1 =>
-- Real
temp_re(0) := resize(matrix_real(0)(I_col), bit_width_adder) - resize(
matrix_real(4)(I_col), bit_width_adder);
temp_re(1) := resize(matrix_imag(2)(I_col), bit_width_adder) - resize(
matrix_imag(6)(I_col), bit_width_adder);
-- MultPart
temp_re(2) := resize(matrix_real(1)(I_col), bit_width_adder) - resize(
matrix_real(3)(I_col), bit_width_adder);
temp_re(3) := resize(matrix_imag(1)(I_col), bit_width_adder) - resize(
matrix_imag(7)(I_col), bit_width_adder);
temp_re(4) := resize(matrix_imag(3)(I_col), bit_width_adder) - resize(
matrix_real(5)(I_col), bit_width_adder);
temp_re(5) := resize(matrix_real(7)(I_col), bit_width_adder) - resize(
matrix_imag(5)(I_col), bit_width_adder);
-- Imag
temp_im(0) := resize(matrix_imag(0)(I_col), bit_width_adder) - resize(
matrix_real(2)(I_col), bit_width_adder);
temp_im(1) := resize(matrix_real(6)(I_col), bit_width_adder) - resize(
matrix_imag(4)(I_col), bit_width_adder);
-- MultPart
temp_im(2) := resize(matrix_imag(1)(I_col), bit_width_adder) - resize(
matrix_real(1)(I_col), bit_width_adder);
temp_im(3) := resize(matrix_real(5)(I_col), bit_width_adder) - resize(
matrix_real(3)(I_col), bit_width_adder);
temp_im(4) := resize(matrix_real(7)(I_col), bit_width_adder) - resize(
matrix_imag(3)(I_col), bit_width_adder);
temp_im(5) := resize(matrix_imag(7)(I_col), bit_width_adder) - resize(
matrix_imag(5)(I_col), bit_width_adder);

-- 3. Zeile aus W
-- 0: (1.00000 + 0.00000i), 1: (0.00000 + 1.00000i), 2: (-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)
when 2 =>
-- Real

```

```

190      temp_re(0) := resize(matrix_real(0)(I_col), bit_width_adder) - resize(
      matrix_real(2)(I_col), bit_width_adder);
      temp_re(1) := resize(matrix_imag(1)(I_col), bit_width_adder) - resize(
      matrix_imag(3)(I_col), bit_width_adder);
192      temp_re(2) := resize(matrix_real(4)(I_col), bit_width_adder) - resize(
      matrix_real(6)(I_col), bit_width_adder);
      temp_re(3) := resize(matrix_imag(5)(I_col), bit_width_adder) - resize(
      matrix_imag(7)(I_col), bit_width_adder);
194      — Imag
      temp_im(0) := resize(matrix_imag(0)(I_col), bit_width_adder) - resize(
      matrix_real(1)(I_col), bit_width_adder);
196      temp_im(1) := resize(matrix_real(3)(I_col), bit_width_adder) - resize(
      matrix_imag(2)(I_col), bit_width_adder);
      temp_im(2) := resize(matrix_imag(4)(I_col), bit_width_adder) - resize(
      matrix_real(5)(I_col), bit_width_adder);
198      temp_im(3) := resize(matrix_real(7)(I_col), bit_width_adder) - resize(
      matrix_imag(6)(I_col), bit_width_adder);

200      — 4. Zeile aus W
      — 0: ( 1.00000 + 0.00000i), 1: (-0.70711 + 0.70711i), 2: (-0.00000 -
      1.00000i), 3: ( 0.70711 + 0.70711i)
202      — 4: (-1.00000 + 0.00000i), 5: ( 0.70711 - 0.70711i), 6: ( 0.00000 +
      1.00000i), 7: (-0.70711 - 0.70711i)
      when 3 =>
204      — Real
      temp_re(0) := resize(matrix_real(0)(I_col), bit_width_adder) - resize(
      matrix_imag(2)(I_col), bit_width_adder);
206      temp_re(1) := resize(matrix_imag(6)(I_col), bit_width_adder) - resize(
      matrix_real(4)(I_col), bit_width_adder);
      — MultPart
208      temp_re(2) := resize(matrix_imag(1)(I_col), bit_width_adder) - resize(
      matrix_real(1)(I_col), bit_width_adder);
      temp_re(3) := resize(matrix_real(3)(I_col), bit_width_adder) - resize(
      matrix_imag(5)(I_col), bit_width_adder);
210      temp_re(4) := resize(matrix_imag(3)(I_col), bit_width_adder) - resize(
      matrix_imag(7)(I_col), bit_width_adder);
      temp_re(5) := resize(matrix_real(5)(I_col), bit_width_adder) - resize(
      matrix_real(7)(I_col), bit_width_adder);
212

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

222      — 5. Zeile
      — 0: (1.00000 + 0.00000i), 1: (-1.00000 + 0.00000i), 2: (1.00000 -
      0.00000i), 3: (-1.00000 + 0.00000i),
224      — 4: (1.00000 - 0.00000i), 5: (-1.00000 + 0.00000i), 6: (1.00000 -

```

```

0.00000i), 7: (-1.00000 + 0.00000i)
  when 4 =>
    — Real
    temp_re(0) := resize(matrix_real(0)(I_col), bit_width_adder) - resize(
matrix_real(1)(I_col), bit_width_adder);
    temp_re(1) := resize(matrix_real(2)(I_col), bit_width_adder) - resize(
matrix_real(3)(I_col), bit_width_adder);
    temp_re(2) := resize(matrix_real(4)(I_col), bit_width_adder) - resize(
matrix_real(5)(I_col), bit_width_adder);
    temp_re(3) := resize(matrix_real(6)(I_col), bit_width_adder) - resize(
matrix_real(7)(I_col), bit_width_adder);
    — Imag
    temp_im(0) := resize(matrix_imag(0)(I_col), bit_width_adder) - resize(
matrix_imag(1)(I_col), bit_width_adder);
    temp_im(1) := resize(matrix_imag(2)(I_col), bit_width_adder) - resize(
matrix_imag(3)(I_col), bit_width_adder);
    temp_im(2) := resize(matrix_imag(4)(I_col), bit_width_adder) - resize(
matrix_imag(5)(I_col), bit_width_adder);
    temp_im(3) := resize(matrix_imag(6)(I_col), bit_width_adder) - resize(
matrix_imag(7)(I_col), bit_width_adder);

    — 6. Zeile
    — 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)
    when 5 =>
      — Real
      temp_re(0) := resize(matrix_real(0)(I_col), bit_width_adder) - resize(
matrix_real(4)(I_col), bit_width_adder);
      temp_re(1) := resize(matrix_imag(2)(I_col), bit_width_adder) - resize(
matrix_imag(6)(I_col), bit_width_adder);
      —MultPart
      temp_re(2) := resize(matrix_real(3)(I_col), bit_width_adder) - resize(
matrix_real(1)(I_col), bit_width_adder);
      temp_re(3) := resize(matrix_real(5)(I_col), bit_width_adder) - resize(
matrix_imag(1)(I_col), bit_width_adder);
      temp_re(4) := resize(matrix_imag(5)(I_col), bit_width_adder) - resize(
matrix_imag(3)(I_col), bit_width_adder);
      temp_re(5) := resize(matrix_imag(7)(I_col), bit_width_adder) - resize(
matrix_real(7)(I_col), bit_width_adder);
      — Imag
      temp_im(0) := resize(matrix_imag(0)(I_col), bit_width_adder) - resize(
matrix_real(2)(I_col), bit_width_adder);
      temp_im(1) := resize(matrix_real(6)(I_col), bit_width_adder) - resize(
matrix_imag(4)(I_col), bit_width_adder);
      —MultPart
      temp_im(2) := resize(matrix_real(1)(I_col), bit_width_adder) - resize(
matrix_imag(1)(I_col), bit_width_adder);
      temp_im(3) := resize(matrix_real(3)(I_col), bit_width_adder) - resize(
matrix_real(5)(I_col), bit_width_adder);
      temp_im(4) := resize(matrix_imag(3)(I_col), bit_width_adder) - resize(
matrix_real(7)(I_col), bit_width_adder);
      temp_im(5) := resize(matrix_imag(5)(I_col), bit_width_adder) - resize(
matrix_imag(7)(I_col), bit_width_adder);

    — 7. Zeile
    — 0: (1.00000 + 0.00000i), 1: (-0.00000 - 1.00000i), 2: (-1.00000 +

```

```

0.00000i), 3: ( 0.00000 + 1.00000i),
260   — 4: (1.00000 - 0.00000i), 5: (-0.00000 - 1.00000i), 6: (-1.00000 +
0.00000i), 7: (-0.00000 + 1.00000i)
    when 6 =>
262   — Real
        temp_re(0) := resize(matrix_real(0)(I_col), bit_width_adder) - resize(
matrix_imag(1)(I_col), bit_width_adder);
264   temp_re(1) := resize(matrix_imag(3)(I_col), bit_width_adder) - resize(
matrix_real(2)(I_col), bit_width_adder);
        temp_re(2) := resize(matrix_real(4)(I_col), bit_width_adder) - resize(
matrix_imag(5)(I_col), bit_width_adder);
266   temp_re(3) := resize(matrix_imag(7)(I_col), bit_width_adder) - resize(
matrix_real(6)(I_col), bit_width_adder);
        — Imag
268   temp_im(0) := resize(matrix_imag(0)(I_col), bit_width_adder) - resize(
matrix_imag(2)(I_col), bit_width_adder);
        temp_im(1) := resize(matrix_real(1)(I_col), bit_width_adder) - resize(
matrix_real(3)(I_col), bit_width_adder);
270   temp_im(2) := resize(matrix_imag(4)(I_col), bit_width_adder) - resize(
matrix_imag(6)(I_col), bit_width_adder);
        temp_im(3) := resize(matrix_real(5)(I_col), bit_width_adder) - resize(
matrix_real(7)(I_col), bit_width_adder);
272
        — 8. Zeile
274   — 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)
276   when 7 =>
        — Real
278   temp_re(0) := resize(matrix_real(0)(I_col), bit_width_adder) - resize(
matrix_imag(2)(I_col), bit_width_adder);
        temp_re(1) := resize(matrix_imag(6)(I_col), bit_width_adder) - resize(
matrix_real(4)(I_col), bit_width_adder);
280   —MultPart
        temp_re(2) := resize(matrix_real(1)(I_col), bit_width_adder) - resize(
matrix_imag(1)(I_col), bit_width_adder);
282   temp_re(3) := resize(matrix_imag(5)(I_col), bit_width_adder) - resize(
matrix_real(3)(I_col), bit_width_adder);
        temp_re(4) := resize(matrix_real(7)(I_col), bit_width_adder) - resize(
matrix_imag(3)(I_col), bit_width_adder);
284   temp_re(5) := resize(matrix_imag(7)(I_col), bit_width_adder) - resize(
matrix_real(5)(I_col), bit_width_adder);
        — Imag
286   temp_im(0) := resize(matrix_imag(0)(I_col), bit_width_adder) - resize(
matrix_imag(4)(I_col), bit_width_adder);
        temp_im(1) := resize(matrix_real(2)(I_col), bit_width_adder) - resize(
matrix_real(6)(I_col), bit_width_adder);
288   —MultPart
        temp_im(2) := resize(matrix_real(1)(I_col), bit_width_adder) - resize(
matrix_imag(3)(I_col), bit_width_adder);
290   temp_im(3) := resize(matrix_imag(1)(I_col), bit_width_adder) - resize(
matrix_real(5)(I_col), bit_width_adder);
        temp_im(4) := resize(matrix_real(3)(I_col), bit_width_adder) - resize(
matrix_imag(5)(I_col), bit_width_adder);
292   temp_im(5) := resize(matrix_imag(7)(I_col), bit_width_adder) - resize(
matrix_real(7)(I_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         — Es wird vor jeder Addition ein Bitshift auf die Summanden angewandt, um
den Wertebereich der Speichervariable beim zurueckschreiben nicht zu
ueberschreiten (1. Mal)

304         — Zeilen 1, 3, 5, 7 (ungerade) aufsummieren (bzw. 0(000XXX), 2(010XXX),
4(100XXX), 6(110XXX) beginnend bei 0)
306         if element(3) = '0' then

308         — Real
310         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);
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);
312         — Imag
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);
314         else
316         — gerade Zeilen aus W
— Real
318         —ConstPart
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);
320         —MultPart
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);
322         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);
— Imag
324         —ConstPart
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);
326         —MultPart
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);
328         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);
    end if;
330
    next_dft_state <= additions_stage2;
332
334    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)
336        — Zusaetzlich wird beim Zuweisen der ungeraden Zeilen an die 1D-DFT-
        Matrix zwei wweitere Male geshiftet.
        — 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
338
        — Zeilen 1, 3, 5, 7 (wie oben)
340        if element(3) = '0' then
342            — 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);
344            — Imag
            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);
346
            — 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
348            — oft geshiftet, aber auch 1x mehr aufaddiert.
            if dft_1d_2d = '0' then
350                dft_1d_real(I_col)(row_col_idx) := resize(temp_re(0)(bit_width_adder
            -1 downto 2), bit_width_extern);
                dft_1d_imag(I_col)(row_col_idx) := resize(temp_im(0)(bit_width_adder
            -1 downto 2), bit_width_extern);
352            else
                result_real(I_col)(row_col_idx) <= resize(temp_re(0)(bit_width_adder
            -1 downto 2), bit_width_extern);
354                result_imag(I_col)(row_col_idx) <= resize(temp_im(0)(bit_width_adder
            -1 downto 2), bit_width_extern);
            end if;
356
            element := element+1;
            element_out <= element;
358
            — naechster Zustand
            next_dft_state <= twiddle_calc;
360
362        else
364            — 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);
366
            — Imag
368            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);

```



```

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

374
    when const_mult => -- dft_state_out = 4
376
      -- Der Zielvektor der Multiplikation ist 26 Bit breit, die beiden
      Multiplikanten sind mit je 13 Bit wie gefordert halb so breit.
378
      -- Zeilen 2, 4, 6, 8 (vergleichbar mit oben)
      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));
382
      next_dft_state <= additions_stage3;
384

    when additions_stage3 => -- dft_state_out = 5
386
      -- 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

      templ4bit_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) := templ4bit_re(bit_width_adder downto 1);
394

      -- Imag
      templ4bit_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) := templ4bit_im(bit_width_adder downto 1);
398

      if dft_1d_2d = '0' then
400        dft_1d_real(I_col)(row_col_idx) := temp_re(0)(bit_width_adder-1 downto 1);
        dft_1d_imag(I_col)(row_col_idx) := temp_im(0)(bit_width_adder-1 downto 1);
402      else
        result_real(I_col)(row_col_idx) <= temp_re(0)(bit_width_adder-1 downto 1);
        result_imag(I_col)(row_col_idx) <= temp_im(0)(bit_width_adder-1 downto 1);
404      end if;

406
      if element = 63 then
408        if dft_1d_2d = '1' then
          next_dft_state <= set_ready_bit;
410
          --report "Bitbreite der Eingangswerte ist " &integer'image(
          bit_width_extern);
          --write(LineBuffer, std_logic_vector(twiddle_coeff(16 downto 16-(
          bit_width_adder-1)))));
          --writeline(output, LineBuffer);
414        else
          next_dft_state <= twiddle_calc;
416        end if;
        dft_1d_2d := not dft_1d_2d;

```

```

418         dft_1d_2d_out <= dft_1d_2d;
         end if;

420

422         element := element+1;
         element_out <= element;

424

426         when set_ready_bit =>
             result_ready <= '1';
428             next_dft_state <= twiddle_calc;

430

         when others => next_dft_state <= twiddle_calc;
432     end case;

434     end process;
end arch;

```

Listing 8.15: Berechnung der 2D-DFT

```

library ieee;
2 use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
4 library work;
  use work.all;
6 use constants.all;
  use datatypes.all;
8
entity dft8optimiert_top is
10 --     port(
--         result_real : out t_2d_array;
12 --         result_imag : out t_2d_array
--     );
14 end entity dft8optimiert_top;

16 architecture arch of dft8optimiert_top is

18     signal nReset      : bit;
     signal clk          : bit;
20     signal input_real   : t_2d_array;
     signal input_imag   : t_2d_array;
22     signal result_real  : t_2d_array;
     signal result_imag  : t_2d_array;
24     signal loaded      : bit;
     signal result_ready  : bit;
26     signal write_done   : bit;
     signal idft          : bit := '0';
28
     signal state_out     : t_dft8_states;
30     signal element_out  : unsigned(5 downto 0);
     signal dft_1d_2d_out : bit;
32

34     component dft8optimiert
        port(
36         clk          : in  bit;
         nReset        : in  bit;

```

```

38         loaded      : in  bit;
         input_real   : in  t_2d_array;
40         input_imag  : in  t_2d_array;
         result_real  : out t_2d_array;
42         result_imag : out t_2d_array;
         result_ready : out bit;
44         idft        : in  bit;
         state_out    : out t_dft8_states;
46         element_out : out unsigned(5 downto 0);
         dft_1d_2d_out : out bit
48     );
end component;

50

52 component read_input_matrix
    port(
54         clk          : in  bit;
         loaded        : out bit;
56         input_real   : out t_2d_array;
         input_imag    : out t_2d_array
58     );
end component;

60

62 component write_results
    port(
64         result_ready : in  bit;
         result_real   : in  t_2d_array;
66         result_imag  : in  t_2d_array;
         write_done    : out bit
68     );
end component;

70

72 begin
    dft : dft8optimiert
74         port map(
            nReset      => nReset,
76             clk        => clk,
            loaded      => loaded,
78             input_real => input_real,
            input_imag  => input_imag,
80             result_real => result_real,
            result_imag => result_imag,
82             result_ready => result_ready,
            idft        => idft,
84             state_out  => state_out,
            element_out  => element_out,
86             dft_1d_2d_out => dft_1d_2d_out
        );
88
    mat : read_input_matrix
90         port map(
            clk          => clk,
92             loaded      => loaded,
            input_real   => input_real,
94             input_imag  => input_imag
        );

```

```

96     write : write_results
97     port map(
100         result_ready => result_ready,
101         result_real  => result_real,
102         result_imag  => result_imag,
103         write_done   => write_done
104     );
105
106     clk    <= not clk after 20 ns;
107     nReset <= '1' after 40 ns;
108 end arch;

```

Listing 8.16: Top-Level-Entität der 2D-DFT

## 8.7 Testumgebung

```

#!/bin/bash
2
matlab_script="binMat2decMat.m"
4
./simulate.sh && matlab -nojvm -nodisplay -nosplash -r $matlab_script
6
stty echo

```

Listing 8.17: Aufruf der Testumgebung, Vergleich von VHDL- und Matlab-Ergebnissen

tlab

```

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="nchhdl.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} \

```

```



```

Listing 8.18: Simulations des VHDL-Quelltextes

```
run 32us
```

Listing 8.19: Dauer der Simulation

```

1 filename_2 = 'InputMatrix_komplex.txt';
  filename_1 = 'Results.txt';
3
5 delimiterIn = ' ';
7
9 bit_width_extern = 13
10
11 Input_bin = importdata(filename_2, delimiterIn);
12 Input_bin_real = Input_bin(:,1:2:end);
13 Input_bin_imag = Input_bin(:,2:2:end);
14
15 Results_vhdl_bin = importdata(filename_1, delimiterIn);
16 Results_vhdl_bin_real = Results_vhdl_bin(:,1:2:end);
17 Results_vhdl_bin_imag = Results_vhdl_bin(:,2:2:end);
18
19 Input_dec_imag = nan(8);
20 Results_vhdl_dec_real = nan(8);
21 Results_vhdl_dec_imag = nan(8);
22 Result_octave_real_1d = nan(8);
23 Result_octave_imag_1d = nan(8);

```

```

23 a=fi(0,1,bit_width_extern,bit_width_extern-2);
25
27 N = 8;
28 for m = 1:N
29     for n = 1:N
30         a.bin=mat2str(Results_vhdl_bin_real(m,n),bit_width_extern);
31         Results_vhdl_dec_real(m,n) = a.double;
32         a.bin=mat2str(Results_vhdl_bin_imag(m,n),bit_width_extern);
33         Results_vhdl_dec_imag(m,n) = a.double;
34
35         a.bin=mat2str(Input_bin_real(m,n),bit_width_extern);
36         Input_dec_real(m,n) = a.double;
37         a.bin=mat2str(Input_bin_imag(m,n),bit_width_extern);
38         Input_dec_imag(m,n) = a.double;
39     end
40 end
41
42 Input_dec=Input_dec_real+1i*Input_dec_imag;
43
44
45 TW=exp(-i*2*pi*[0:7]'.*[0:7]/8);
46
47
48
49
50 %Result_octave_1d=TW*Input_dec;
51 %Result_octave_real_1d=real(Result_octave_1d.)/16
52 %Result_octave_imag_1d=imag(Result_octave_1d)
53
54 Result_octave=TW*Input_dec*TW. ';
55 Result_octave=Result_octave./256;
56
57 Results_vhdl_dec_real
58 Result_octave_real=real(Result_octave)
59
60 Result_octave_imag=imag(Result_octave);
61 Results_vhdl_dec_imag;
62
63 diff_real=Result_octave_real-Results_vhdl_dec_real
64 diff_imag=Result_octave_imag-Results_vhdl_dec_imag;
65
66 quit

```

Listing 8.20: Berechnung der Differenzen der DFT in Matlab und VHDL