

10.04.2018

1 Einleitung

1.1 Motivation

Sensorarray beschreiben = warum ein Array, Vorteile?

1.2 Stand der Technik

Der verwendete Prozess ist mit $350\text{ }\mu\text{m}$ im Vergleich zu modernen Prozessen mit beispielsweise
14-20 nm Strukturbreite um die Größenordnung 10^4 größer. Entsprechend handelt es sich um einen
relativ alten Prozess.

Kurze Beschreibung zu Standardzellen.

!!!

1.3 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 auf relevante Anteile 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.

↓ Aufgabestellung einarbeiten!

10.04.2018

2 Grundlagen

Um einen guten Ausgangspunkt für spätere Erläuterungen zu haben, sollen ~~an~~ an dieser Stelle die wesentlichen Grundlagen zusammengefasst werden.

2.1 Binäre Zahlendarstellung von Festkommazahlen

Im Rahmen dieses Projekts wird von Ein- sowie Ausgangswerten mit einer Genauigkeit von 12 Bit ausgegangen. Basierend auf älteren Sensoren wird von Werten im Bereich von $-2 < z < 2$ ausgegangen. Aus diesem Grund müssen sowohl ein Ganzahlanteil, sowie Nachkommastellen repräsentiert werden können. Wie dies gelingt, wird in den nächsten Abschnitten gezeigt. Hierfür werden Festkommazahlen verwendet, aufgrund der Rechenoperationen haben diese dennoch unterschiedlich viele Vor- sowie Nachkommastellen.

2.1.1 Integer-Zahl im 1er-Komplement

Bei der Interpretation des Bitvektors als Integerwert im Einerkomplement werden die Bits anhand ihrer Position im Bitvektor gewichtet, wobei ~~as~~ niedrigwertigste 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, wobei eine 0 für eine positive Zahl steht. Entsprechend besagt die 1, dass die Zahl negativ ist. Dies hat zur Folge, dass es eine positive und eine negative Null und somit eine Doppeldeutigkeit gibt. Des Weiteren wird ein LSB an Auflösung verschenkt. Der Wertebereich erstreckt sich von $-2^{\text{MSB}-1} + 1$ LSB bis $2^{\text{MSB}-1} - 1$ 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.

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.

2.1.2 Integer-Zahl 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^{-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 $2^{MSB-1} - 1$ 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 das Resultat muss abschließend 1 LSB addiert werden. J

Vorteil 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.

2.1.3 Darstellung dualer Zahlen im SQ-Format

Im SQ-Format werden Zahlen als vorzeichenbehafteter Quotient (signed quotient) dargestellt. Wie beim 2er-Komplement entscheidet das höchstwertigste Bit, ob es sich um eine positive oder negative Zahl handelt. In Abbildung 2.1 ist exemplarisch die Interpretation von Dualzahlen im SQ3-Format, also für vier Bit, zu sehen.

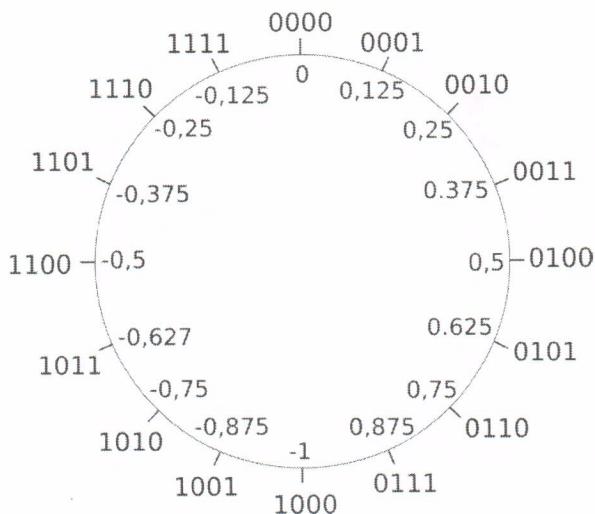


Abbildung 2.1: Interpretation von Dualzahlen im SQ3-Format

Der darstellbare Zahlenbereich liegt hier bei $-1 \leq z < 1$. Benötigt werden Zahlen im Bereich von etwa ± 2 , weshalb ein Vorkommabit benötigt wird. Da 12 Bit zur Verfügung stehen, von denen eins für das Vorzeichen und ein weiteres für eine Vorkommastelle verwendet werden, bleiben 10 Bits für die Nachkommazahlen übrig. Die Aufteilung der Bits wird über die Bezeichnung S1Q10 definiert. Da für den Quotient 10 Bit zur Verfügung stehen, beträgt die maximale Auflösung $1 \text{ LSB} = 2^{-10} = 1024^{-1} = 9,765625 \cdot 10^{-4}$. Der Wertebereich liegt in diesem Fall liegt bei -2 bis 1.999 023 438. e

Für die Addition oder Multiplikation zweier Zahlen müssen beide einerseits die selbe Bitbreite und andererseits das gleiche Darstellungsformat besitzen.

t

2.1.4 Numerisch bedingte Ungenauigkeiten

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

1
2 3
1 2

+/-

Da diese Arbeit den Schwerpunkt in der Aufwandsabschätzung einer Chipimplementation einer 2D-DFT auf einem Application Specific Integrated Circuit, dt.: *Anwendungsspezifischer Integrierter Schaltkreis (ASIC)* hat, ist diese Problematik kein Gegenstand dieser Arbeit und wird an dieser Stelle nur in Grundzügen erwähnt.

2.2 Mathematische Grundlagen

Zu den mathematischen Grundlagen werden die komplexe Multiplikation sowie die Matrixmultiplikation gezählt, welche nachfolgend kurz behandelt werden. Auf die Fouriereihenentwicklung sowie insbesondere die Fouriertransformation und ihre diskrete Variante wird im Anschluss detaillierter eingegangen, da sie elementarer Bestandteil dieser Arbeit sind. Da auch die diskrete Kosinustransformation als mögliche Transformationsart im Raum stand, um in den Bildbereich zu gelangen, wird diese ebenfalls kurz aufgegriffen.

d

2.2.1 Komplexe Multiplikation

Im allgemeinen Fall müssen gemäß Gl. (2.1) bei der komplexen Multiplikation vier einfache Multiplikation sowie zwei Additionen durchgeführt werden.

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

2.2.2 Matrixmultiplikation

Um nachfolgende Abschnitte besser erörtern zu können, soll zunächst die Matrixmultiplikation besprochen werden. Wie in Abbildung 2.2 verdeutlicht, wird Element (i, j) der Ergebnismatrix dadurch berechnet, dass die Elemente (i, k) einer Zeile der 1. Matrix mit den Elementen (k, j)

aus der zweiten Matrix multipliziert und die Werte aufsummiert werden. i und j sind für die Berechnung eines Elements konstant, während k über alle Elemente einer Zeile bzw. Spalte läuft.

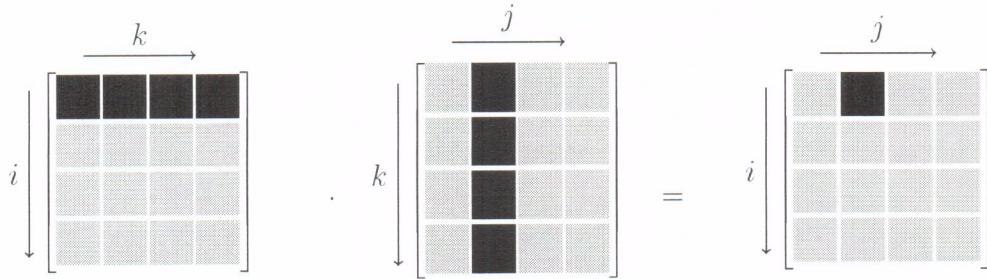


Abbildung 2.2: Veranschaulichung der Matrixmultiplikation

2.3 Fourierreihenentwicklung

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

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

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

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

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

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

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

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

und zusammengefasst ergibt sich in (Gl. 2.7) der komplexe Zeiger, der eine Rotation im Gegenuhrzeigersinn auf dem Einheitskreis beschreibt. In Abbildung 2.3 dies zusätzlich noch grafisch dargestellt.

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

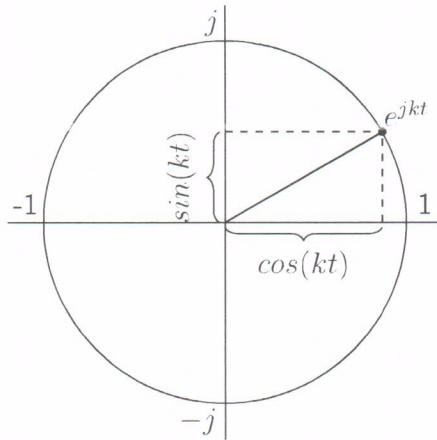


Abbildung 2.3: Einheitskreis, Zusammensetzung des komplexen Zeigers aus Sinus und Kosinus

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

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

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

2.4 Fouriertransformation

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

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

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

Wenn Sinus- und Kosinusfunktionen wie in Gl. (2.5) und (2.6) als Exponentialfunktion

geschrieben werden, können sie zu einer komplexen Exponentialfunktion zusammengefasst werden. Hieraus lässt sich ableiten, dass das Spektrum, also $X(f)$ komplexwertig sein muss.

Für Signalformen wie etwa ein Rechteck haben entsprechend sehr viele dieser Frequenzbeiträge. 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, mit ihr erhält man das Spektrum eines Zeitsignals. Eine Vertiefung dieses umfangreichen Gebiets der Fourier-Analyse findet sich u.a. in ...

S

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

Ergänza

2.4.1 Diskrete Fouriertransformation (DFT)

Die Diskrete Fouriertransformation (DFT) ist die zeit- und wertdiskrete Variante der Fouriertransformation, die statt von $-\infty$ bis ∞ über einen Vektor von N Werten, also von 0 bis $N-1$ läuft. Dies hat zur Folge, dass sich ihr Frequenzspektrum periodisch nach N Werten wiederholt.

Da es sich um eine endliche Anzahl diskreter Werte handelt, geht das Integral aus Gleichung (2.10) in die Summe aus Gleichung (2.11) über.

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

uw

Statt zeitlich versetzter Daten werden beim Sensorarray die Daten von mehreren Sensoren gleichzeitig erfasst. Da das Sensorarray zweidimensional ist, ergibt sich an Stelle eines Vektors so eine Matrix. Weil die Werte gleichzeitig erfasst werden und diese verschiedene Koordinaten repräsentieren, muss hier von Orts- anstatt von Zeitwerten gesprochen werden. Von der Transformation ins Frequenzspektrum spricht man wiederum bei Zeitwerten, da das Spektrum die Frequenzen darstellt, aus denen das Zeitsignal zusammengesetzt ist. Da bei der eben beschriebenen Datenerfassung Ortsdaten transformiert werden, spricht man hier allgemeiner von einer Transformation in den Bildbereich.

d

In dieser Arbeit werden statt Zeit- bzw. Ortsbereich respektive Frequenzbereich und Bildverarbeitung häufig auch die Begriffe Ein- und Ausgangsvektor bzw. -matrix verwendet.

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

2.4.2 Summen- und Matrzenschreibweise der DFT

1D-DFT

Die DFT findet wie bereits erwähnt üblicherweise Anwendung, um vom Zeit- in den Frequenzbereich zu gelangen.

$$X^*[m] = \frac{1}{N} \cdot \sum_{n=0}^{N-1} x[n] \cdot e^{-\frac{j2\pi mn}{N}} \quad (2.11)$$

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

Mit Gleichung (2.12) werden 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.

$$\sum_{m=0}^{N-1} \sum_{n=0}^{N-1} e^{-\frac{j2\pi mn}{N}} = W \quad (2.12)$$

Somit gilt:

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

In Matlab kann die Twiddlefaktormatrix mit

$$W = e^{-\frac{j2\pi}{N} \cdot [0:N-1]' \cdot [0:N-1]} \quad (2.14)$$

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

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.

$$\begin{aligned} X[u, v] &= \frac{1}{N} \sum_{n=0}^{N-1} X^*[m] \cdot e^{-\frac{j2\pi mn}{N}} \\ &= \frac{1}{MN} \sum_{m=0}^{M-1} \left(\sum_{n=0}^{N-1} f(m, n) \cdot e^{-\frac{j2\pi mn}{N}} \right) \cdot e^{-\frac{j2\pi mn}{M}} \end{aligned} \quad (2.15)$$

Auch hier lässt sich die Berechnung in Matrzenschreibweise darstellen:

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

Die Gleichungen (2.13) und (2.16) werden wesentlicher Bestandteil der Umsetzung der 2D-DFT sein.

Wie in Gleichung (2.16) beschrieben, kann die 2D-DFT als "doppelte" Matrizenmultiplikation geschrieben werden. Es wird also erst die 1D-DFT berechnet und die sich daraus ergebende Matrix X^* (Abb. 2.17) wird anschließend mit der Twiddlefaktor-Matrix W multipliziert. Man könnte es auch als zweite 1D-DFT betrachten, bei der Twiddlefaktor-Matrix und Eingangsma-
trix vertauscht sind.

Veranschaulicht wird dies in den Abbildungen 2.17 und 2.18.

$$\begin{bmatrix} W \\ \vdots \\ \vdots \end{bmatrix} \cdot \begin{bmatrix} x \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} X^* \\ \vdots \\ \vdots \end{bmatrix} \quad (2.17)$$

$$\begin{bmatrix} X^* \\ \vdots \\ \vdots \end{bmatrix} \cdot \begin{bmatrix} W \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} X \\ \vdots \\ \vdots \end{bmatrix} \quad (2.18)$$

2.4.3 2D-DFT mit reellen Eingangswerten

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

Es wäre jedoch auch möglich, das komplexe Ausgangssignal der 1D-DFT als zwei von einander unabhängige rein reelle Eingangssignale der 2D-DFTs zu betrachten und später wieder zusammenzusetzen. Gleiches gilt dann natürlich auch für ein komplexes Eingangssignal, welches ebenfalls in zwei von einander unabhängigen DFTs transformiert werden kann. Da bei dieser Umsetzung kein Imaginärteil in die Berechnung der Ergebnisse einfließt, hat sie den Vorteil, dass aus Symmetriegründen die Hälfte der Multiplikationen eingespart werden können. Hierbei ist es erforderlich, dass der Imaginärteil der gespiegelten Ergebnisse negiert wird. Abbildung 2.5 zeigt die redundanten Werte der DFT. Es müssen bei der 8x8-DFT also statt 16 nur 8 Multiplikationen mit realem Multiplikand und komplexen Multiplikator erfolgen.

Wie bereits beschrieben, lässt sich dieses Verfahren auch für komplexe Eingangssignale, deren Real- und Imaginärteil separaten von einander mit der DFT transformiert werden, anwenden. Anschließend müssen die Ergebnisse zusammen gesetzt werden. Wie dies geschieht ist der Abbildung 2.4 zu entnehmen. Die Abbildung stellt die schematische Berechnung der 2D-DFT

1
2
3

eines reellen Eingangssignals dar. Um die 2D-DFT eines komplexen Eingangssignals zu berechnen, muss entweder eine identische Einheit für den Imaginärteil vorhanden sein oder noch mehr zeitlich versetzt berechnet werden. Die Ergebnisse beider 2D-DFTs müssen identisch zusammengefasst werden, wie es zum Abschluss der einzelnen 2D-DFTs geschehen muss.

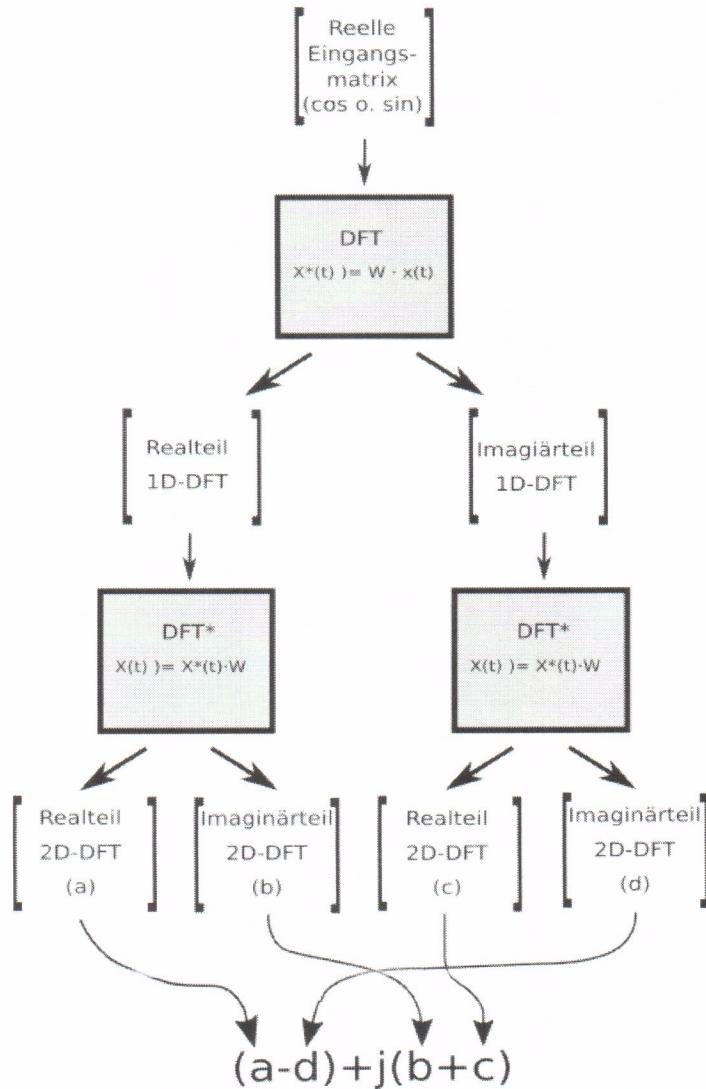


Abbildung 2.4: Veranschaulichung der Berechnung der DFT mit reellen Eingangswerten

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

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

(a) Realteil

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

(b) negierter Imaginärteil

Abbildung 2.5: Redundante Werte der spaltenweisen DFT einer 8x8-Matrix. Der Imaginärteil der redundanten Werte hat den selben Betrag mit negiertem Vorzeichen.

2.4.4 Berechnung der Diskreten Fouriertransformation mittels FFT

Die Mathematiker Cooley und Tukey haben einen Algorithmus entwickelt und im Jahr 1965 veröffentlicht, mit dem sich die DFT mit vergleichsweise wenig Multiplikationen und somit deutlich schneller als bei der allgemeinen DFT berechnen lässt. Das Verfahren wird als Fast Fouriertransformation (FFT) bezeichnet. Grundlage ist, dass sich eine DFT in kleinere Teil-DFTs aufspalten lässt, welche durch Ausnutzen von Symmetrieeigenschaften in der Summe weniger Koeffizienten haben. Üblich ist die Radix-2 FFT, Ausgangspunkt ist also eine DFT mit 2 Eingangswerten. Da mit jeder weiteren Teil-DFT sich die Anzahl der Eingangswerte verdoppelt, eignet sich diese Methode nur für Eingangsvektoren der Größe 2^n . Dieser vermeindliche Nachteil lässt sich durch Auffüllen des Eingangsvektors mit Nullen (Zeropadding) eliminieren. Dies hat zur Folge, dass die Größe des Ausgangsvektors immer eine Potenz von zwei ist. Abbildung 2.6 illustriert dies anhand eines Eingangsvektors mit acht Werten. Um diesen Algorithmus anwenden zu können ist es erforderlich, dass die Werte im Eingangsvektor in umgekehrte Bitreihenfolge getauscht werden (bitreversed order). Dies geschieht nach dem Muster, dass die Indizes der Eingangswerte, wie üblich bei 0 beginnend, binär dargestellt werden. Nun wird die Reihenfolge der Bits getauscht. Auf diese Weise tauschen bei einem 8-Rit Vektor die Elemente 2 und 5 sowie 4 und 7 ihre Position.

Aus Gleichung (2.12) ist bekannt, dass die Variablen der Twiddlefaktorberechnung die Indizes der Eingangs- sowie Ausgangsvektoren sind. Hieraus lässt sich bereits erkennen, dass die gesamte Twiddlefaktormatrix N verschiedene komplexe Werte enthält. Dies wird auch aus Abbildung 3.1 aus Abschnitt 3.1.2 am Beispiel für N=8 ersichtlich. Darüber hinaus lässt sich erkennen, dass die komplexen Zeiger den Einheitskreis in N Bereiche mit einem Winkel von $\frac{2\pi}{N}$ unterteilen. Bekannt ist ebenfalls, dass der erste Wert immer die 1 ist. Daraus ergibt sich bei

einer DFT mit 2 Eingangswerten die Twiddlefaktoren 1 und -1 , sodass eine Multiplikation entfällt.

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

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

Wie gezeigt wurde, werden nur zwei komplexe Multiplikationen benötigt. Eine Abschätzung der benötigten komplexen Multiplikationen erhält man mit der Gleichung (2.19):

$$\frac{N}{2} \log_2(N) = \frac{8}{2} \cdot 3 = 12 \quad (2.19)$$

Insbesondere bei größeren FFTs ist die relative Abweichung bedeutend geringer.

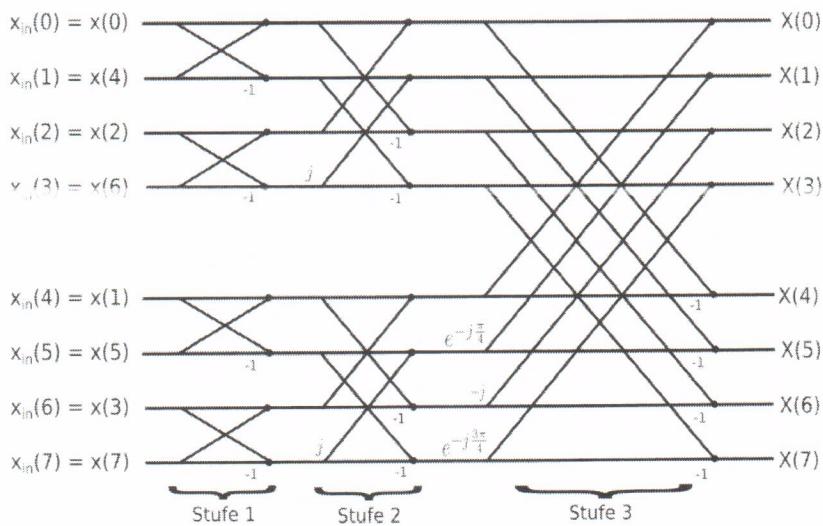


Abbildung 2.6: Berechnungsschema der DFT mit 8 Eingangswerten nach dem Butterfly-Verfahren

2.4.5 Inverse DFT

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

?. hier fehlt etwas!

$$x[n] = \frac{1}{N} \sum_{n=0}^{N-1} X^*[m] \cdot e^{\frac{j2\pi mn}{N}} \quad (2.20)$$

beschrieben. Gleichung (2.20) ist bis auf die Drehrichtung des komplexen Zeigers und die den v den Vertauschten Ein- und Ausgangsvektoren identisch zu Gleichung (2.11).

2.5 Diskrete Kosinus Transformation (DCT)

2.5.1 Verwendung der DCT

Die DCT findet häufig in der Bildverarbeitung Anwendung.

Text

2.5.2 Berechnung der DCT

Für die Berechnung der DCT gibt es verschiedene Varianten, welche sich in der Symmetrie der Ergebnismatrix unterscheiden. (Stimmt das wirklich? was sonst?)

Darüber hinaus wird in der Bildverarbeitung häufig die 1. Zeile der Twiddlefaktormatrix mit dem Faktor $\frac{1}{\sqrt{2}}$, sowie die gesamte Matrix mit $\sqrt{\frac{2}{N}}$, N = Anzahl Elemente in einer Zeile bzw. Spalte, multipliziert.

Da es hier um eine Aufwandsabschätzung geht, wird sich auf die in der Bildverarbeitung gängigste Variante jedoch ohne die skalierenden Faktoren beschränkt. Diese berechnet sich zu

$$X^*[k] = \sum_{n=0}^{N-1} x[n] \cos \left[\frac{\pi k}{N} \left(n + \frac{1}{2} \right) \right] \quad \text{für } k = 0, \dots, N-1 \quad (2.21)$$

Die Twiddlefaktormatrix kann in Matlab mit

$$W = \cos \left(\frac{\pi}{N} \cdot \left([0:N-1]' * ([0:N-1] + \frac{1}{2}) \right) \right) \quad (2.22)$$

berechnet werden. Da die Diskrete Cosinus Transformation (DCT) anders als die DFT nur auf Kosinusfunktionen und nicht aus einer Kombination aus Sinus und Kosinus, liefert sie rein reellwertige Werte.

e f

3 Analyse

Im diesem Kapitel werden zunächst die DFT und die DCT in verschiedenen Größen einander gegenübergestellt und eine Entscheidung darüber getroffen, welche sich besser dafür eignet, auf einem ASIC implementiert zu werden. Hierbei spielen in erster Linie die Anzahl unterschiedlicher Faktoren eine Rolle, da für identische Faktoren nur eine Multiplikationseinheit nötig ist. Gleiche Faktoren gehen also mit einer kleineren Chipfläche einher, was zusammen mit der schnellen Berechnung, also geringe Zahl benötigter Takte, die beiden Hauptziele bei der Chipimplementierung darstellen. Als interessante Kandidaten wurden primär die Matrizen mit den Größen 8x8, 9x9 und 15x15 ausgewählt. Die 8x8-Matrix hat die selbe Anzahl der Sensoren wie das derzeitige Demo-Array, so dass die Eingangswerte direkt transformiert werden können. Die beiden anderen haben aufgrund ihrer ungeraden Zahl ihren Mittelpunkt zwischen den mittleren Sensorelementen, was für die weitere Verarbeitung des transformierten Signals von Bedeutung ist. Die Matrix der Dimension 15x15 lässt sich durch Interpolation der Daten errechnen, während es für die 9x9 bisher keine Überlegungen gibt, wie sie errechnet werden könnte. Die 12x12 sowie die 16x16 werden zum besseren Einordnen der Bewertungen ebenfalls betrachtet. Darüber hinaus ist aus Abschnitt 2.4.4 bekannt, dass die FFT auf 2^n Elementen basiert und es sich hierbei um ein sehr schnelles und effizientes Verfahren handelt.

Im zweiten Schritt wird untersucht, wie die 8x8-DFT, welche als Favorit aus der ersten Betrachtung herausgegangen ist, optimiert werden kann.

3.1 Bewertung verschiedener DFT- und DCT-Größen

In diesem Abschnitt sollen Erkenntnisse gewonnen werden, auf denen basierend später die Wahl der Transformation und die Größe ihrer Matrix getroffen werden kann. Die Bewertung berücksichtigt wie bereits angedeutet die beiden Eigenschaften Anzahl verschiedener Faktoren und die gesamte Anzahl an Faktoren, wobei die erst genannte größeren Einfluss auf eine negative Bewertung hat, da sich gleiche Faktoren mit Hilfe des Distributivgesetzes ausklammern lassen. Bei den Faktoren wird zwischen solchen unterschieden, die als trivial erachtet werden, da sie nur eine Addition bedürfen (± 1), zusätzlich zur Addition nur eine Division durch 2 erfolgt ($\pm 0,5$) oder gar keine Berechnung nötig ist (0) und solchen, die als nicht trivial betrachtet werden müssen, da eine Multiplikation unumgänglich ist (beispielsweise $\frac{\sqrt{2}}{2}$). Begründet werden kann dies mit dem dualen Zahlensystem, da eine Multiplikation mit als trivial eingestuften Werten kein komplexes Schaltnetz erfordert. Um z.B. einen Wert durch 2 zu teilen, erfolgt im dualen einfach ein Bitshift um eins nach rechts.

System, Wertsystem....

A

3.1.1 Bewertung verschiedener DCT-Größen

In Tabelle 3.1 ist die Gegenüberstellung der genannten Größen zu sehen. Für die Bewertung wurde das Matlab-Skript aus Anhang 8.1 geschrieben. Ersichtlich ist, dass die Anzahl verschiedener nicht trivialer Werte etwa der Wurzel aus der Anzahl aller Werte ist. Dies bedeutet im Umkehrschluss, dass im Schnitt jede Zeile einen neuen Faktor einführt. Die Summe nicht trivialer Werte weist bei allen Matrizen mehr als 50% auf.

Tabelle 3.1: Bewertung der DCT-Twiddlefaktor-Matrizen

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

3.1.2 Bewertung verschiedener DFT-Größen

In der Tabelle 3.2 werden die DFT-Matrizen einander gegenüber gestellt. Anders als die DCT haben die Twiddlefaktormatrix und deshalb auch das Ergebnis der DFT einen Real- und einen Imaginärteil. Die Beurteilung basiert auf dem Matlab-Skript aus Anhang 8.2. Wie zu sehen ist, schneiden vor allem die 8x8- und die 12x12-DFT gut ab. Da letztere nur zum Vergleich mit aufgenommen wurde, ist die 8x8-DFT der klare Favorit, welcher im folgenden Abschnitt genauer betrachtet werden soll.

Tabelle 3.2: Bewertung der DFT-Twiddlefaktor-Matrizen

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

3.2 Genauere Betrachtung der 8x8-DFT

In Grafik 3.1 sind die Twiddlefaktoren der 8x8-DFT im Einheitskreis dargestellt. Betragsmäßig treten die Werte 0, 1 und $\sqrt{2}/2$ auf. Gemäß der obigen Definition für nicht triviale Werte zählt ausschließlich der letztgenannte zu diesen. Eine besondere Eigenschaft ist, dass für nicht triviale Multiplikationen Real- und Imaginärteil zumindest vom Betrag her identisch sind. Dies liegt daran, dass der Einheitskreis in acht Teile geteilt wird und für beispielsweise $\frac{2\pi}{8} = \frac{\pi}{4}$ der Sinus- und Kosinuswert identisch sind. Darüber hinaus 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. 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 könnten. Die andere Hälfte müsste lediglich negiert werden. Deshalb kann das berechnete Verhältnis von 3 in Tabelle 3.2 als deutlich höher angenommen werden.

Anfangs wurde in Betracht gezogen, das 1er-Komplement zu verwenden, da hierbei zwei betragsmäßig identische Zahlen sich nur durch ihr höchstwertigstes Bit unterscheiden. Auf diese Weise könnte das ~~selbe~~ Resultat für den Imaginär- wie für den Realteil verwendet werden, das Vorzeichen würde sich über eine einfache XOR-Verknüpfung beider MSB der Multiplikanden ergeben. Diesem Vorteil steht jedoch eine aufwändigere Subtraktion (bzw. Addition negativer Zahlen) gegenüber. Der zusätzliche Aufwand entspricht etwa dem der Bildung des 2er-Komplements. Aus diesem Grund und da das 2er-Komplement deutlich verbreiteter ist sowie weitere Vorteile bringt wie beispielsweise keine Doppeldeutigkeit durch eine negative Null hat, wurde sich hierfür entschieden.

In späteren Analysen konnte festgestellt werden, dass sowohl für den Real- als auch den Imaginärteil gleichviele Multiplikationen mit positiven wie mit negativen Faktoren durchgeführt werden müssen. Dies lässt sich anhand des Einheitskreises in Abb. 3.1 und der Abbildung 3.2 nachvollziehen. Aus Abschnitt 2.2.2 ist bekannt, dass bei einer Matrixmultiplikation Elemente multipliziert und anschließend die Ergebnisse aufsummiert. Da es das Kommutativgesetz erlaubt, die berechneten Ergebnisse auch in einer anderen Reihenfolge zu addieren,

In Abbildung 3.2 sind zur besseren 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 Laufindizes (n) des Frequenzbereichs zu sehen.

In Abbildung 3.3 ist zu sehen, dass die 2., 4., 6., und 8. Zeile je vier nicht triviale und komplexe Faktoren enthält. Darüber hinaus ist ersichtlich, dass für komplexe Eingangswerte in den genannten Zeilen 12 und in den übrigen 8 Multiplikationen erfolgen müssen. Dies kann anhand der Gleichungen (2.1) und (3.1) nachvollzogen werden. Das lässt sich ausnutzen, um keine Negationen der Eingangs- und Zwischenwerte durchführen zu müssen.

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

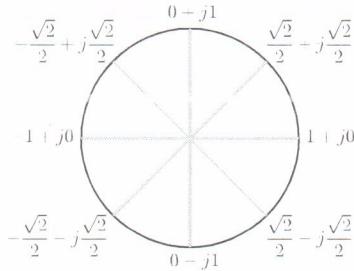
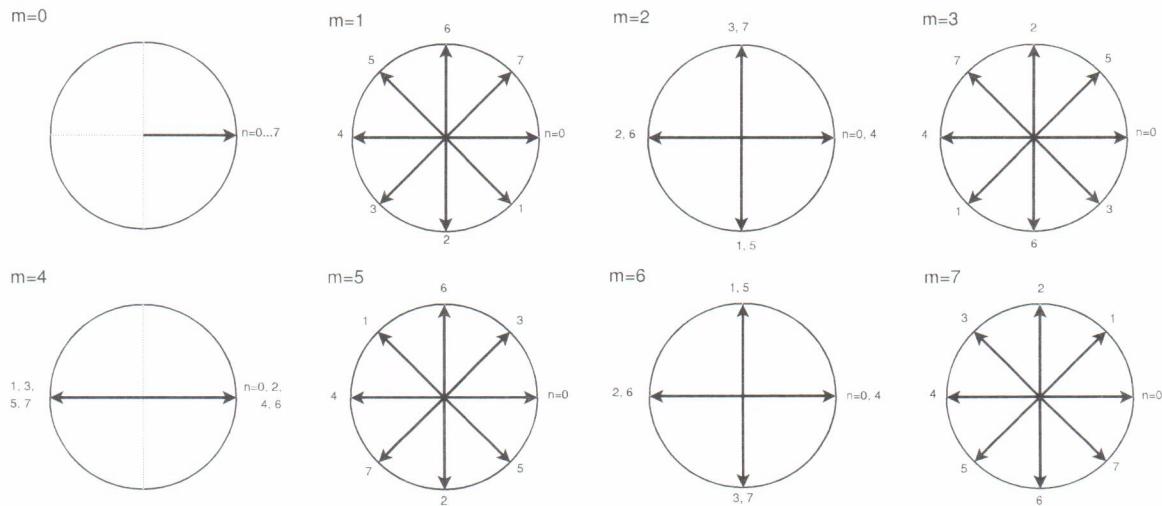


Abbildung 3.1: Einheitskreis mit relevanten Werten der 8x8-DFT

Darüber hinaus minimiert sich bei geschickter Anordnung das Risiko eines Überlaufs. Wie in Abschnitt 2.1.4 erwähnt, ist dies nicht Gegenstand dieser Arbeit, weshalb der Einfachheit wegen zur Sicherheit dennoch nach jeder Addition oder Subtraktion das Ergebnis durch einen Bitshift halbiert wird. Es sei an dieser Stelle lediglich angemerkt, dass über die Eingangswerte die Annahme getroffen werden kann, dass aufeinanderfolgende Werte das selbe Vorzeichen haben. Dies ließe sich ausnutzen, um noch weiter die Wahrscheinlichkeit zu reduzieren, dass es zu einem Überlauf kommt.

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

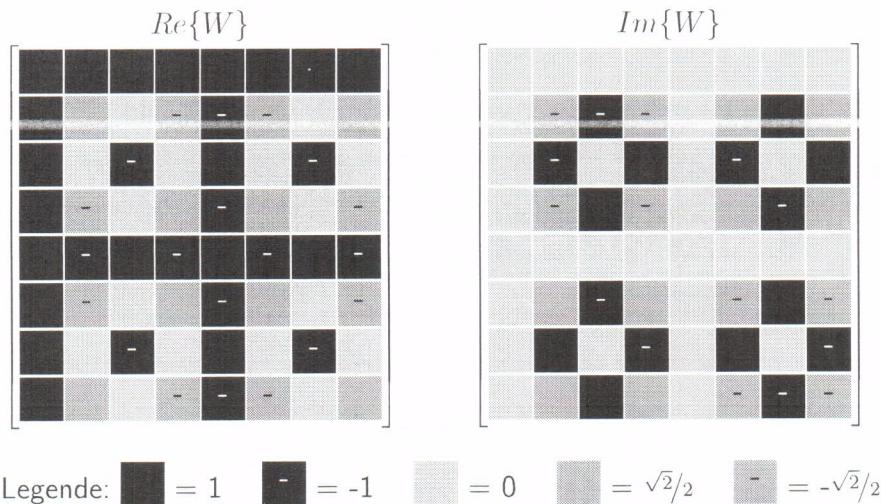


Abbildung 3.3: Matrix-Darstellung der 8x8-DFT-Twiddlefaktoren, aufgeteilt nach Real- und Imaginärteil

Sowohl ~~die~~ der Abbildung 3.2 als auch insbesondere ~~der~~ die Darstellung 3.3 lassen sich sehr gut die Symmetrien erkennen, die diese Twiddlefaktormatrix so vorteilhaft machen. ~~die die f~~

3.3 Entscheidung über Größe und Art der Transformation

Noch nicht fertig!

Sowohl die DCT als auch die DFT finden häufig in der Bildverarbeitung Anwendung. Der Vorteil der DCT gegenüber der DFT ist, dass sie rein reelle Ergebniswerte liefert. Ihr großer Nachteil zeigt sich u.a. insbesondere deutlich bei den 8x8-Matrizen, da sich hier

~~A~~ nicht trivial darstellbare Zahlen der DCT einem einzigen bei der 8x8-DFT gegenüber stehen. ~~R~~ ~~U~~
Auch wenn bei der DFT mit der Berechnung des imaginären Teils zusätzlicher Implementierungsaufwand hinzukommt, wird davon ausgegangen, dass dieser geringer ist, als alle x Multiplikationen umzusetzen. Ebenso ist die Annahme, dass der Platzbedarf auf einem Chip in einer ähnlichen Größenordnung liegt, da auf der einen Seite der zusätzliche Speicherbedarf für eine weitere Matrix den x Konstantenmultiplizierer-Schaltnetzen gegenüber stehen.

Es ist nicht geklärt, welche Berechnung für eine Weiterverarbeitung sinnvoller ist. Dies heraus zu finden ist jedoch nicht Bestandteil der Aufgabenstellung dieser Arbeit. An dieser Stelle sollen lediglich Vor- und Nachteile zusammengetragen werden, die eine Entscheidung rechtfertigen.

Ein Einsatzszenario der Transformationen ist die Filterung von Rauschen und anderen Störgrößen. Hierfür ist die DFT gut geeignet.

Da es bei dieser Arbeit vor allem um die Aufwandsabschätzung einer optimierten Matrizenmultiplikation zur Vorverarbeitung der Sensordaten geht, welche als Ausgangspunkt für eine

finale Implementation dient, und es sich hier um keine endgültige Entscheidung handelt, ist die DFT gut geeignet.

Tabelle 3.3: Gegenüberstellung der Vor- und Nachteile von DCT und DFT

Eigenschaft	Vorteil	Nachteil
Imaginärteil Vorhanden	DCT	DFT
Anzahl Multiplikationen	DFT	DCT
Platzbedarf	-	-

3.4 Abschätzung des Rechenaufwands

3.4.1 Gegenüberstellung von der Konstantenmultiplikation und der Bildung des 2er-Komplements

Unter diesem Punkt sollen die Konstantenmultiplikation und die Bildung des 2er-Komplements unter Aspekten der benötigten Zeit und des benötigten Platzes auf einem Chip betrachtet werden. Um einen Eindruck hiervon zu erhalten, werden im Kapitel Entwurf in den Abschnitten 4.4.1 und 4.4.2 jeweils die Schaltnetze gezeigt. Wie dort erläutert, lässt sich anhand dieser feststellen, dass es bei dieser Art der Implementierung keinen zeitlichen Gewinn gibt, da beide kritischen Pfade etwa gleich lang sind. Für die knapp $\frac{1}{4}$ mehr Gatter bei der Multiplikation ist auch ein größerer Verdrahtungsaufwand erforderlich, sodass die Konstantenmultiplizierer auf einem Chip eine etwas größere Fläche beanspruchen. Da es sich hier insgesamt aber um sehr wenige Gatter handelt, wirkt sich dies erst bei sehr vielen Instanzen aus. Es kann an dieser Stelle deshalb festgehalten werden, dass dieser Unterschied nicht als entscheidend geltend gemacht werden kann.

3.4.2 Gegenüberstellung von reellen und komplexen Eingangswerten

Die Sensormatrix liefert für jedes Sensorelement einen Sinus- und einen Kosinuswert. Diese können für die Berechnung der DFT zu einer komplexen Zahl zusammengefasst werden. Auf diese Weise lässt sich die Berechnung mathematisch kompakter schreiben.

In Tabelle 3.4 ist eine Auflistung der für die Berechnung veranschlagten Takte für die Multiplikation einer beliebigen Matrix mit der Twiddlefaktormatrix für die 8x8-DFT zu sehen. Grundlage ist, dass in einem Takt Summanden paarweise aufaddiert werden und in einer Variablen zwischengespeichert werden. Dieses Verfahren kann auch als Baumstruktur aufgefasst werden. Wie das Aussummieren erfolgt, kann in Abschnitt 4.7 detaillierter nachgelesen werden.

Wie in Abschnitt 4.4 gezeigt wird, kann die Multiplikation mit einer Konstanten innerhalb eines Taktes mit einem Schaltnetz erfolgen. Anders als bei der komplexen Multiplikation mit der Twiddlefaktormatrix sind bei der getrennten Berechnung ungleich viele positive und negative Faktoren je Zeile vorhanden, sodass zu diesem Zeitpunkt davon ausgegangen werden muss, dass eine Negation mancher Werte erforderlich sein wird. In Abschnitt 4.4.2 wird gezeigt, dass

der kritische Pfad der Negierung sogar etwas länger als beim Konstantenmultiplizierer ist. Um keine zu langen Signal- und Gatterlaufzeiten erwerben, sollte hierfür unbedingt ein eigener Takt eingeplant werden. Dadurch relativiert sich der zeitliche Gewinn allerdings etwas.

Tabelle 3.4: Takte für die komplexe DFT

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

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

Wie ein Vergleich der Gleichungen (2.1) und (3.1) zeigt, entfallen die Hälfte der Multiplikationen, wenn die Eingangswerte in Real- und Imaginärteil getrennt werden. Wenn die Eingangswerte rein reell sind, kommen beispielsweise keine j^2 -Komponenten zustande, welche auf die reellen Elemente aufaddiert werden müssten. Aus diesem Grund müssen weniger Werte aufsummiert werden, wie sich in Tabelle 3.5 zeigt.

Tabelle 3.5: Takte für die reelle DFT am Beispiel der reellen Ausgangsmatrix

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

Aus Abschnitt 2.4.3 ist bekannt, dass die letzten drei Zeilen direkt oder negiert aus den Zeilen 2-4 übernommen werden können. Die Takte der ~~6-8~~ Zeilen sind deshalb in der Tabelle (3.5) grau hinterlegt. Gegenüber der komplexen Matrix ergeben sich hier statt 256 Takten $(3+4+2+4+3) \cdot 8 = 128$ Takte. Der Imaginärteil errechnet sich noch schneller, da die ~~5.~~ und ~~6.~~ Zeile keinen Beitrag leisten und auch hier die Zeilen 2-4 in diesem Fall nach einer Negation

~~fünf~~1 s.u. +
erste

sechste bis acht

die Werte der letzten 3 Zeilen ergeben. So ergeben sich dort $(3+2+3) \cdot 8 = 64$ Takte. Vermutlich müssen an dieser Stelle wieder Takte für das Negieren eingeplant werden. Da beide parallel berechnet werden, sind die hierfür benötigten Takte sozusagen frei verfügbar.

Interessant ist dieser Ansatz dann, wenn einerseits die Recheneinheit so klein wie irgend möglich gehalten werden soll und andererseits die Berechnung noch schneller erfolgen muss. Abbildung 2.4 zeigt, dass im Vergleich zur komplexen Berechnung der 2D-DFT voraussichtlich 3x so viel Speicher für Zwischenwerte vorhanden sein muss. Insgesamt übersteigt so der Flächenbedarf der gesamten Einheit der der komplexen Variante. Auch die Leitungen um den Speicher anzubinden dürfen nicht vernachlässigt werden.

3.4.3 Direkte Multiplikation zweier 8x8 Matrizen

Die in Abschnitt 2.2.2 erläuterte Matrixmultiplikation bedarf bei einer 8x8 Matrix je Ergebnis der Ausgangsmatrix 8 Multiplikationen. Für die $8 \cdot 8 = 64$ Elemente werden deshalb 512 Multiplikationen benötigt. Da es sich sowohl bei den Eingangswerten als auch bei der Twiddlefaktormatrix um komplexe Zahlen handelt, sind, wie in Abschnitt 2.2.1 beschrieben, insgesamt $512 \cdot 4 = 2048$ Multiplikationen nötig.

Sollte sich dazu entschieden werden, die Sinus- und Kosinusanteile separat zu berechnen, um ein rein reelles Eingangssignal weiter zu verarbeiten, sind, wie in Abschnitt 2.4.3 hergeleitet, knapp die Hälfte der Multiplikationen unnötig. In Abbildung 2.5 ist zu sehen, dass von den 64 Ergebniswerten nur 40 berechnet werden müssen. Da die Eingangswerte zwar rein reell, die Twiddlefaktormatrix aber komplex ist, verdoppelt sich die Anzahl der Multiplikationen. Somit müssen für die gesamten 64 Werte $40 \cdot 8 \cdot 2 = 640$ Multiplikationen durchgeführt werden.

Im komplexen Fall verdoppelt sich für die 2D-DFT schlicht die Anzahl der reellen Multiplikationen und liegt somit bei 4096. Im reellen Fall müssen, wie in Abbildung 2.4 gezeigt, der Real- sowie der Imaginärteil separat mit der Twiddlefaktormatrix multipliziert werden. So ergeben sich alles in allem $640 \cdot 3 \cdot 2 = 3840$ reelle Multiplikationen. Diese Zahl liegt nur geringfügig unterhalb der komplexen Berechnung.

Hierbei wird von einer Twiddlefaktormatrix mit 64 komplexen Werten ausgegangen. In Wirklichkeit sind es nur 16, die übrigen erfordern überhaupt keine Multiplikation, da entweder der Real- oder der Imaginärteil 0 ist. Da dies aber Bestandteil der optimierten Matrixmultiplikation ist, wird an dieser Stelle nicht weiter darauf eingegangen. Später werden nur die komplexen Varianten verglichen. Dies wird als ausreichend erachtet, da aufgrund der hier und in Abschnitt 2.4.3 angedeutete deutlich erhöhte Bedarf an Takten die reelle Matrixmultiplikation nicht von Interesse ist.

3.4.4 Optimierte 8x8 DFT als Matrixmultiplikation

Die Twiddlefaktormatrix lässt sich unter verschiedenen Aspekten betrachten. Zunächst sticht die erste Zeile hervor, da sie im Realteil nur aus positiven Einsen und im Imaginärteil nur aus Nullen besteht. Mit der fünften Zeile verhält es sich ähnlich. Anders ist hier, dass sich positive und negative Einsen abwechseln. In die nächste Gruppe können die dritte und die siebte Zeile zusammengefasst werden. Beide haben gemeinsam, dass sie so wie die erste und fünfte nur

Einsen und Nullen als Faktoren haben. Im Unterschied zu den vorigen können hier aber auch die Imaginärteile eine positive bzw. negative Eins haben. Entsprechend ist dann der Realteil Null. Hier müssen zur Berechnung des Ergebnisses also auch Imaginärteile der Eingangsmatrix mit einbezogen werden. Für die vier bisher betrachteten Zeilen gilt, dass zur Berechnung eines Elements der Ergebnismatrix ausschließlich Additionen oder Subtraktionen erforderlich sind. Da die komplexen Zeiger nur aus entweder einem Real- oder einem Imaginärteil bestehen, beschränkt sich die Anzahl der Berechnungen der Elemente der Ausgangsmatrix auf acht Additionen bzw. Subtraktionen.

Die übrigen vier Zeilen haben alle gemein, dass die Hälfte der Faktoren sowohl einen Real- als auch einen Imaginärteil besitzen. Für diese vier Faktoren sind deshalb jeweils insgesamt vier Multiplikationen nötig.

Für den Realteil bedeutet dies, dass die Elemente der ersten Zeile der Ergebnismatrix durch ~~Aufsummieren~~ aller Elemente der korrespondierenden Spalte der Eingangsmatrix berechnet werden. Dies hat zur Folge, dass ein sehr großer Wert entstehen kann, welcher ~~Maßgebend~~ für die Anzahl der Vorkommabits sein kann. Wegen der Null im Imaginärteil der ersten Zeile der Twiddlefaktormatrix sind auch alle Imaginärteile der ersten Spalte der Ausgangsmatrix Null. Bezogen auf den Imaginärteil gilt das ~~gleiche~~ auch für die fünfte Zeile der Twiddlefaktormatrix. Da sich beim Realteil der fünften Zeile der Twiddlefaktormatrix der fünften Zeile positive und negative Einsen abwechseln, sind hier auch keinerlei Multiplikationen nötig.

Aus der anfänglichen Implementation bei der alle nicht trivialen Werte einer Berechnung die entweder mit $+\frac{\sqrt{2}}{2}$ oder $-\frac{\sqrt{2}}{2}$ multipliziert wurden, ~~einzelnd~~ berechnet werden, wird mittels des Distributivgesetzes der gemeinsame Faktor ausgeklammert, sodass nur noch jeweils eine Multiplikation erforderlich ist.

Bei den weiteren Zeilen sind hingegen die Zahlen zur Hälfte positiv und zur anderen negativ. Außerdem enthalten die geraden Zeilen den Faktor $\pm\frac{\sqrt{2}}{2}$. Dies lässt sich ausnutzen, um die Anzahl der Multiplikationen zu reduzieren. Zunächst können die ~~- - -~~ ergänzen!

Für jede gerade Zeile der DFT ist jeweils für den Real- und den Imaginärteil eine Multiplikation nötig, so dass sich insgesamt acht Multiplikationen ergeben

3.4.5 Gegenüberstellung von Butterfly-Algorithmus und optimierter Matrixmultiplikation

Die DFT wurde als Matrixmultiplikation implementiert, um die gewonnenen Erkenntnisse auch auf andere Dimensionen als 2^n , insbesondere ungerade, übertragen zu können. Zu einem frühen Zeitpunkt der Überlegungen für diese 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 sowie die Größe der Matrix über eine globale Deklaration definierbar sein. 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 wurde bereits in

Grafik (2.6) gezeigt. Sie stellt eine effiziente Berechnung der DFT dar, in Abschnitt (3.4.4) konnte gezeigt werden, dass sich beide nur unwesentlich im Rechenaufwand unterscheiden.

3.5 Kompromiss aus benötigter Chipfläche und Genauigkeit des Ergebnisses

Dieser Abschnitt passt hier nicht so richtig hin! Aber wo sonst?

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 ~~Bit~~ 1 ist. Im Mittel ist dies bei der Hälfte der Additionen der Fall. In 50% aller Fälle wird also der Wert um ein halbes LSB zu viel verringert. Bei der Multiplikation verdoppelt sich sogar die resultierende Bitbreite. Da mit dem vollständigen 13-Bit-Vektor nach der Addition weitergerechnet wird, muss die Konstante ebenfalls in 13 Bit hinterlegt sein. Deshalb hat das Ergebnis 26 Bit, von denen für die weitere Berechnung wieder nur 12 übernommen werden. In den Abbildungen (4.5) und (4.6) wird das hier beschriebene Vorgehen veranschaulicht. Bei diesem Verfahren kommt es unweigerlich zur Akkumulation von Fehlern.

Da für die Berechnung einer Zahl der 1D-DFT je nach Zeile entweder 8 oder 12 Werte akkumuliert sowie 0 bis 4 Werte multipliziert werden und 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.

Mit jeder Addition kommt 1 Bit dazu. So werden aus 12 Bit bis zur Multiplikation 15 ($12 + \log_2(8)$), 8 = Anzahl der Zahlen die mit $\frac{\sqrt{2}}{2}$ multipliziert werden müssen. Bei der Multiplikation verdoppelt sich der Wert, also 30 und eine letzte Addition macht 31. Beim zweiten Durchlauf werden es so $(31+3)\cdot 2 + 1 = 69$ Bit.

⇒ Anhand eines Simulationsbeispiels zeigen, dass die mit VHDL berechneten Werte immer kleiner als die in Matlab berechneten sind.

notwendig
1

B7

= -

⇒ Hinzufüge von Rauschen (stochastik)
vermindert die Verschleppung von Rundungsfehlern!
Eine Art „Dithering“ verhindert dies.

10.04.2018

4 Entwurf

4.1 Interpretation binärer Zahlen

Matlab fi

immer 10 Nachkommastellen, außer bei Multiplikation

NC Sim, nur Integerdarstellung möglich, bei Vektoren sogar nur positiv

4.2 Entwicklungsstufen

4.2.1 Multiplikation

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

4.2.2 Addierer

CLA, RC, in einem Takt

4.2.3 Konstantenmultiplikation

Dieser Punkt muss irgendwie mit der Implementierung des Konstantenmultiplizierers zusammengeführt werden.

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

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

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

Der Berechnung aus Gleichung (4.1) kann entnommen werden, dass die Abweichung weit unter einem Prozent liegt.

$$\frac{100}{\sqrt{2}} \cdot 0,70703125 = 99,989\% \quad (4.1)$$

¹⁾, getrennt durch Kommas, der Decimal Wert

4.2.4 1D-DFT mit Integer-Werten

4.2.5 2D-DFT mit Integer-Werten

4.2.6 2D-DFT mit Werten SQ-Format

4.2.7 Zusammenhang von DFT und IDFT bei der Matrixmultiplikation

Durch die umgekehrte Drehrichtung des komplexen Zeigers in Gleichung (2.20) werden in der Matrzenschreibweise die Zeilen 2 und 8, 3 und 7 sowie 4 und 6 vertauscht. Nachvollziehen lässt sich das gut anhand der Grafik (3.1). Verdeutlicht wird das ✓ Vorgehen in Abbildung 4.1.

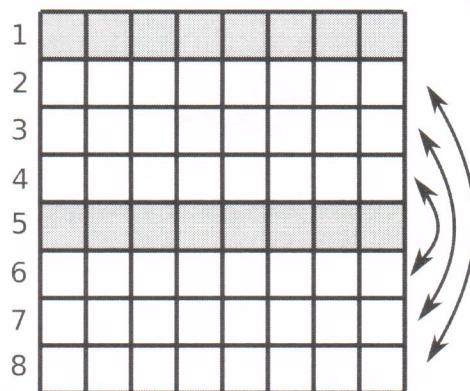


Abbildung 4.1: Um von der DFT zur IDFT zu kommen, müssen bei der Matrixmultiplikation die Zeilen 2 und 8, 3 und 7 sowie 4 und 6 der Twiddlefaktormatrix vertauscht werden

4.3 Test der Matrixmultiplikation

Unter anderem weil NC Sim bzw. dessen Unterprogramm SimVision zur Anzeige von Signalverläufen (Waveform) nur Integer darstellen kann und bei als Vektor gebündelten Signalen diese nicht einmal als vorzeichenbehaftet (signed), wurde der Einfachheit halber zunächst 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. Deshalb konnte an dieser Stelle noch auf den Bitschift 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.

$$\frac{\sqrt{2}}{2}_{10} = 0001011010100_2 \text{ in S2Q10, als Integer betrachtet jedoch } 724_{10}.$$

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!)

deck

4.4 Implementierung des Konstantenmultiplizierers

Anfangs wurde angenommen, dass Multiplikationen mit den Twiddlefaktoren ± 1 und $\pm \frac{\sqrt{2}}{2}$ durchgeführt werden müssen. Dass bei einer optimierten 8x8-DFT wegen des expliziten ausprogrammierens der Berechnungen die Multiplikation mit ± 1 wegfällt, wurde recht schnell klar. Erst bei genauer Betrachtung der Twiddlefaktor-Matrix ~~f~~ 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 Inverter zu verzichten. Weiter wird auch nur die Multiplikation mit $+\frac{\sqrt{2}}{2}$ benötigt.

+ A
f ,

4.4.1 Syntheseergebnis eines 13 Bit Konstantenmultiplizierers

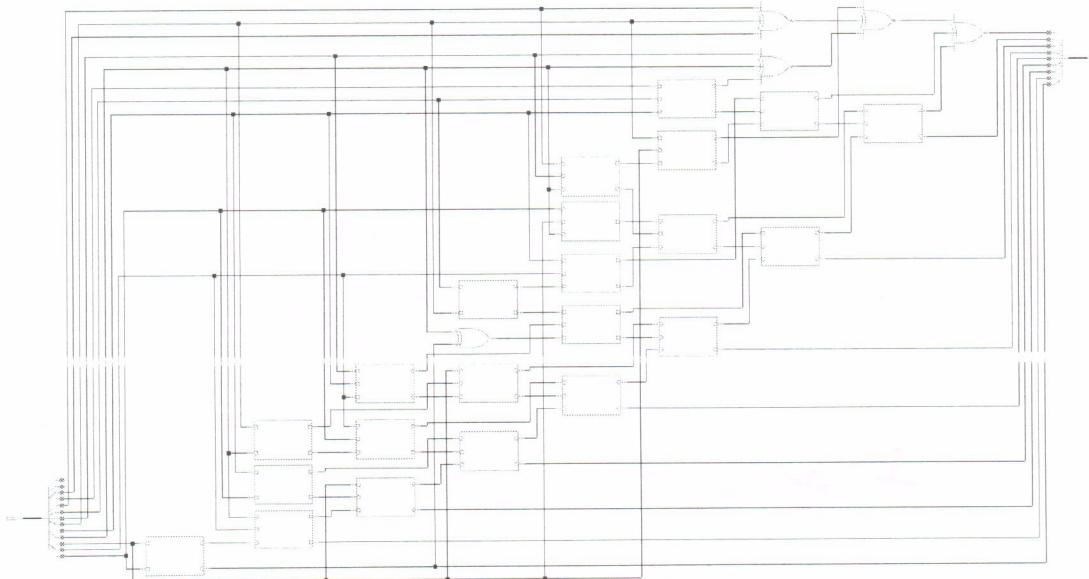


Abbildung 4.2: 13 Bit Konstantenmultiplizierer für $\frac{\sqrt{2}}{2} = 0.70711 \simeq 0.70703125 = 0001011010100_2$ in Encounter; Eingang links, Ausgang rechts

Der vollständige Gate-Report befindet sich in Abschnitt 8.3 auf Seite 51

Tabelle 4.1: Vergleich Konstanten- mit regulärem Multiplizierer

	Konstantenmultiplizierer	regulärer Multiplizierer
Gatter	27	175
Fläche (Prozess: 350nm)	6612 μm^2	23 261 μm^2

4.4.2 Syntheseergebnis für die Bildung des Zweierkomplements eines 13 Bit Vektors

Zum Vergleich mit dem Konstantenmultiplizierer aus Abb. 4.2 soll in Abb. 4.3 die nicht explizit implementierte, aber in Abschnitt 3.4.2 erwähnte Negierung von Zahlen gezeigt werden.

+

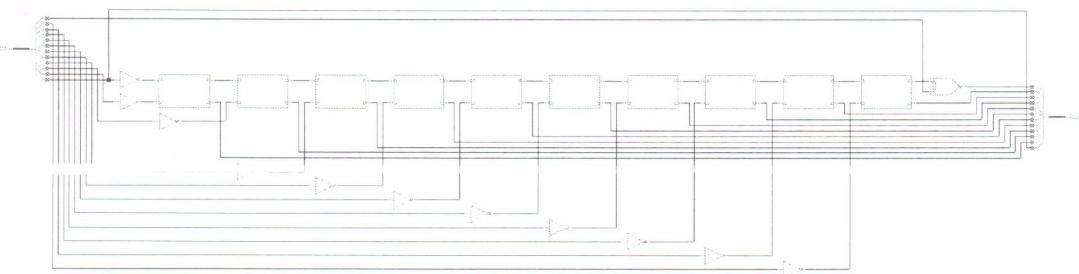


Abbildung 4.3: Netzliste einer Einheit zur Bildung des 2er-Komplements eines 13 Bit Vektors; Eingang links, Ausgang rechts

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

g u

4.5 Entwickeln der 2D-DFT in VHDL

Ziel ist es, die gleiche DFT-Einheit für beide DFTs zu verwenden

Zähler für 64 Werte kann als 6 Bit Vektor realisiert werden, der bei 63 einen Überlauf hat und wieder bei 0 anfängt.

Vorderen 3 Bit sind die der Zeile, die hinteren für die Spalte.

Das dritte Bit von vorne sagt einem, ob es eine gerade oder ungerade Zeile ist.

Die in Gleichung (2.16) beschriebene Berechnung der 2D-DFT lässt sich auch wie folgt schreiben:

l
h p ä h z e

$$\begin{aligned}
 X &= W \cdot x \cdot W \\
 &= (x^T \cdot W)^T \cdot W \\
 &= X^* \cdot W \\
 &= ((x \cdot W)^T \cdot W)^T \\
 &= (X^{*T} \cdot W)^T
 \end{aligned}
 \quad (4.2) \quad (4.3)$$

In Matlab muss hierfür entweder die Funktion `transpose()` oder `.'` verwendet werden. Letzteres muss elementweise angewandt werden, da das Apostroph alleine die komplex konjugiert Transponierte bildet.

Die alternativen Schreibweisen der 2D-DFT haben den Vorteil, dass in beiden Fällen die Eingangsmatrix auf der linken Seite steht. Möglich ist dies, da die Twiddlefaktormatrix identisch mit ihrer Transponierten ist. Dass nun in den Gleichungen (4.2) und (4.3) sowohl die Eingangs- als auch die 1D-DFT-Matrix links steht, ist eine wichtige Voraussetzung dafür, dass mit der selben Recheneinheit mit der die 1D-DFT berechnet wird auch die 2D-DFT berechnet werden kann. Die zweite Voraussetzung ist das Transponieren einer Matrix. Diese lässt sich durch spaltenweises Abspeichern und zeilenweises Auslesen der Ergebnismatrix realisieren. Hierfür ist es lediglich notwendig, die beiden Indizes, welche ein Matrixelement ansprechen, beim Speichern getauscht werden. Nun sind nun alle Voraussetzungen erfüllt, um beide Berechnungen mit der selben Einheit durch zu führen. In Grafik (4.4) ist das hier **Beschriebene** veranschaulicht. (Auf diese Weise wird die direkte Weiterverarbeitung von Werten denkbar.)

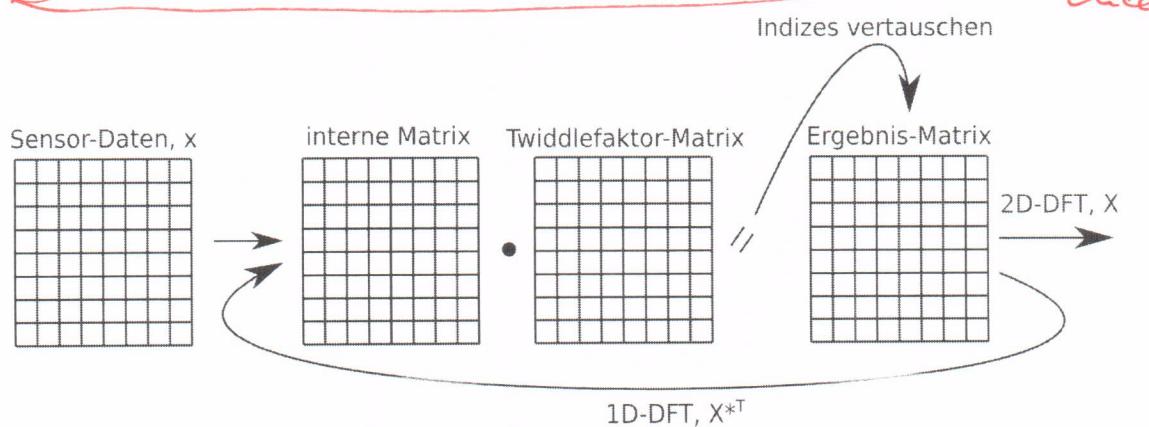


Abbildung 4.4: Darstellung der Berechnung der 2D-DFT aus Gleichung (4.3)

~~Frage~~ neue Seite?

4.6 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 $64 \cdot 2 \cdot 12$ Bit = 1536 Bit = 1,5kBit = 192 Byte an Speicher eingespart werden. Wie sich im Laufe der Entwicklung gezeigt hat, lässt sich das nicht nutzen. Das liegt daran, dass dazu übergegangen wurde, immer nur ein Element zur Zeit berechnet wird und die bereits errechneten demnach zwischengespeichert werden müssen. Dieser Ansatz wurde verfolgt, da der Entwicklungsaufwand in VHDL für die spaltenweise Berechnung der Ausgangswerte einfacher umzusetzen war und es zunächst nur um die mathematische Umsetzung und nicht um die Platzeffizienz auf einem Chip ging.

Unklar war zu diesem Zeitpunkt noch, wie der Speicher realisiert werden soll. In der finalen Variante des Chips soll es einen Random Access Memory (RAM) geben, der als zentraler Speicher von allen Komponenten genutzt wird. Da die Entwicklung im Projekt noch nicht soweit fortgeschritten ist und dies nicht zu den Aufgaben der vorliegenden Arbeit gehört, wurde auf das Speichern in lokalen Speicherzellen ausgewichen, welche als Variable oder Signal im VHDL-Code definiert und von der Software als Flip-Flop synthetisiert werden.

zu berechnen

e

4.7 Berechnungsschema der geraden und ungeraden Zeilen

In Abbildung (4.5) ist die Berechnung der ungeraden Zeilen am Beispiel der ersten zu sehen.

$$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$$

Takt				Bit
1	$\underbrace{a_{k0} + a_{k1}}$	$\underbrace{a_{k2} + a_{k3}}$	$\underbrace{a_{k4} + a_{k5}}$	12
	\Downarrow	\Downarrow	\Downarrow	
2	$\underbrace{\sum_{-1_1} + \sum_{-1_2}}$	$\underbrace{\sum_{-1_3} + \sum_{-1_4}}$		13
	\Downarrow	\Downarrow		
3	$\underbrace{\sum_{-2_1} + \sum_{-2_2}}$			12
	\Downarrow			
		\sum_{-3_1}		13
				12

Abbildung 4.5: Vorgehensweise der Akkumulation der ungeraden Spalten der Eingangswerte

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

Die Berechnung der geraden Zeilen wird in Abbildung (4.6) am Beispiel der zweiten Zeile gezeigt

	$a_0 - x_1 + x_0 - b_2 + x_2 - x_3 + a_4 - x_5 + x_4 - b_6 + x_6 - x_7$						
Takt							Bit
1	$\underbrace{a_0 - x_1}_{\Downarrow}$	$\underbrace{x_0 - b_2}_{\Downarrow}$	$\underbrace{x_2 - x_3}_{\Downarrow}$	$\underbrace{a_4 - x_5}_{\Downarrow}$	$\underbrace{x_4 - b_6}_{\Downarrow}$	$\underbrace{x_6 - x_7}_{\Downarrow}$	12
2	$\underbrace{\sum_{1_1} + \sum_{1_2}}_{\Downarrow}$		$\underbrace{\sum_{1_3} + \sum_{1_4}}_{\Downarrow}$		$\underbrace{\sum_{1_5} + \sum_{1_6}}_{\Downarrow}$		13
3	$\underbrace{\sum_{2_1}}_{\Downarrow}$	+	$\underbrace{\sum_{2_2}}_{\Downarrow}$				12
						\Downarrow	13
			\sum_{3_1}				13
			\Downarrow				13
4		$\frac{\sqrt{2}}{2}$				\Downarrow	13
		\Downarrow					26
5	$\underbrace{\sum_{4_1}}_{\Downarrow}$	+		$\underbrace{\sum_{2_3}}_{\Downarrow}$			12
						\Downarrow	13
			\sum_{5_1}				12

Abbildung 4.6: Vorgehensweise der Akkumulation der geraden Spalten der Eingangswerte

Auch hier ist der linken Spalte die Anzahl der benötigten Takte zu entnehmen. In diesem Fall werden 5 Takte für die Berechnungen benötigt. Diese setzen sich zusammen aus 1. Takt für Additionen bzw. Subtraktionen, 2.-3. sowie 5. Takt für das Aufsummieren und der 4. Takt für die Multiplikationen.

Wie rechts am Rand zu sehen, ergibt sich durch die Addition eine Bitbreitenerweiterung um 1 bzw. bei der Multiplikation eine Verdoppelung. Bei einer früheren Implementierung, die nur die 1D-DFT beherrschte, wurde zumindest die Erweiterung bei der Addition umgesetzt. Da bei der 2D-DFT die selbe Recheneinheit genutzt werden soll, wurde in Absprache mit dem ISAR-Team entschieden, dass die Summanden vor jeder Summation durch einen Bitshift nach rechts halbiert werden. Auf diese Weise hat ein Additionsergebnis immer 13 Bit Breite. Durch den Bitshift kann das Resultat der 1D-DFT direkt als Eingang für die 2D-DFT verwendet werden.

Zu bedenken gilt es bei einem Bitshift, dass das Ergebnis mit jedem Mal eine Division durch 2 erfährt. Bei hintereinander erfolgenden Bitshifts wird demnach durch 2^{N_B} geteilt, wobei N_B die Anzahl der Bitshifts ist. Den beiden obigen Darstellungen der Summationen kann entnommen werden, dass, um ein Überlaufen des Bitvektors zu vermeiden es nötig ist, drei respektive vier Bitshifts durch zu führen. Wie bereits erläutert erfolgt bei den ungeraden Zeilen abschließend ein doppelter Bitshift. Auf diese Weise ergibt sich für die 1D-DFT, dass das Ergebnis um den Faktor 16 kleiner ist, als beispielsweise bei der Berechnung mit Matlab.

Da bei bei dem zweiten Durchlauf, um die 2D-DFT zu berechnen, ebenfalls durch 16 geteilt wird, ergibt sich insgesamt eine Division durch $2^{2 \cdot 4} = 256$.

4.7.1 Erwartete Anzahl benötigter Takte

Aus den Abbildungen 4.5 und 4.6 können die Takte die zur Berechnung der 1D- bzw. 2D-DFT benötigt werden abgeleitet werden.

Für ungeraden Zeilen sind je Element 3 Takte nötig und mit 8 Elementen pro Zeile und 4 ungeraden Zeilen errechnen sich so $3 \cdot 8 \cdot 4 = 96$ Takte. Analog errechnet sich für die ungeraden Zeilen mit je 5 Takten pro Element $5 \cdot 8 \cdot 4 = 160$ Takte. In der Summe ergeben sich so aus mit dem somit das + operations $96 + 160 = 256$ Takte für die 1D-DFT. Da die 2D-DFT ohne Takte fürs Umspeichern oder ähnliches operations sofort im Anschluss berechnet werden kann, verdoppelt sich die Anzahl der Takte auf 512 für die vollständige Berechnung.

4.8 Schema der Zustandsfolge

test test

Text!
~~Text!~~

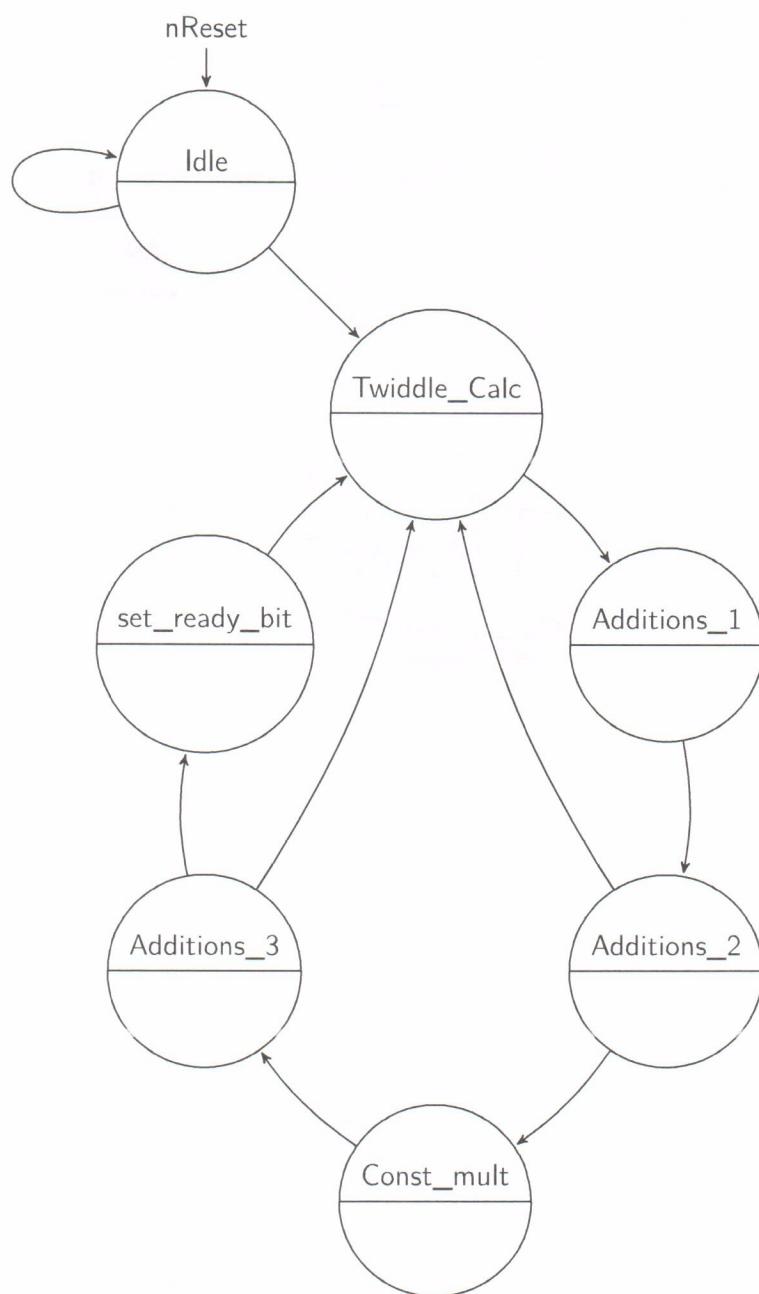
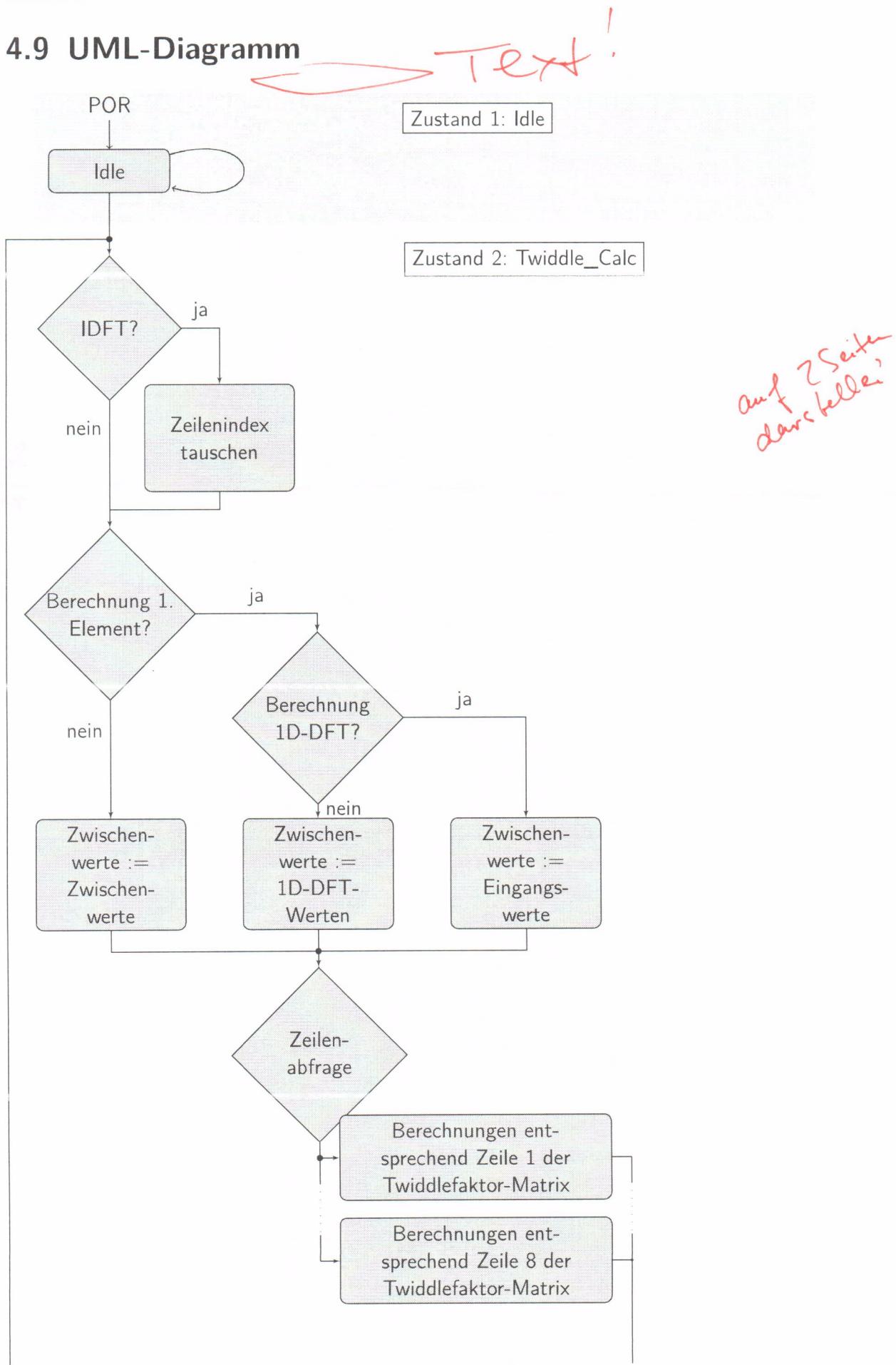
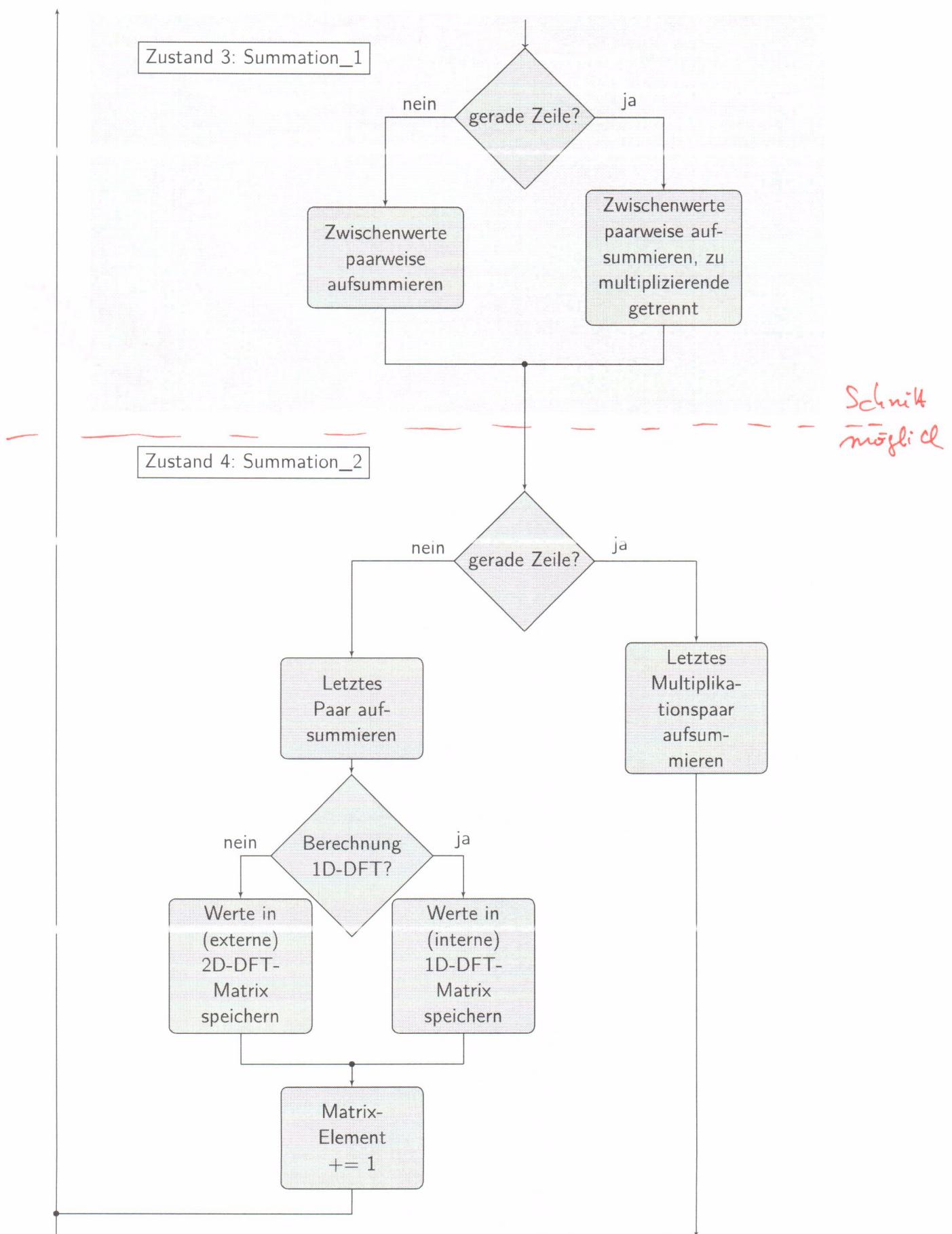
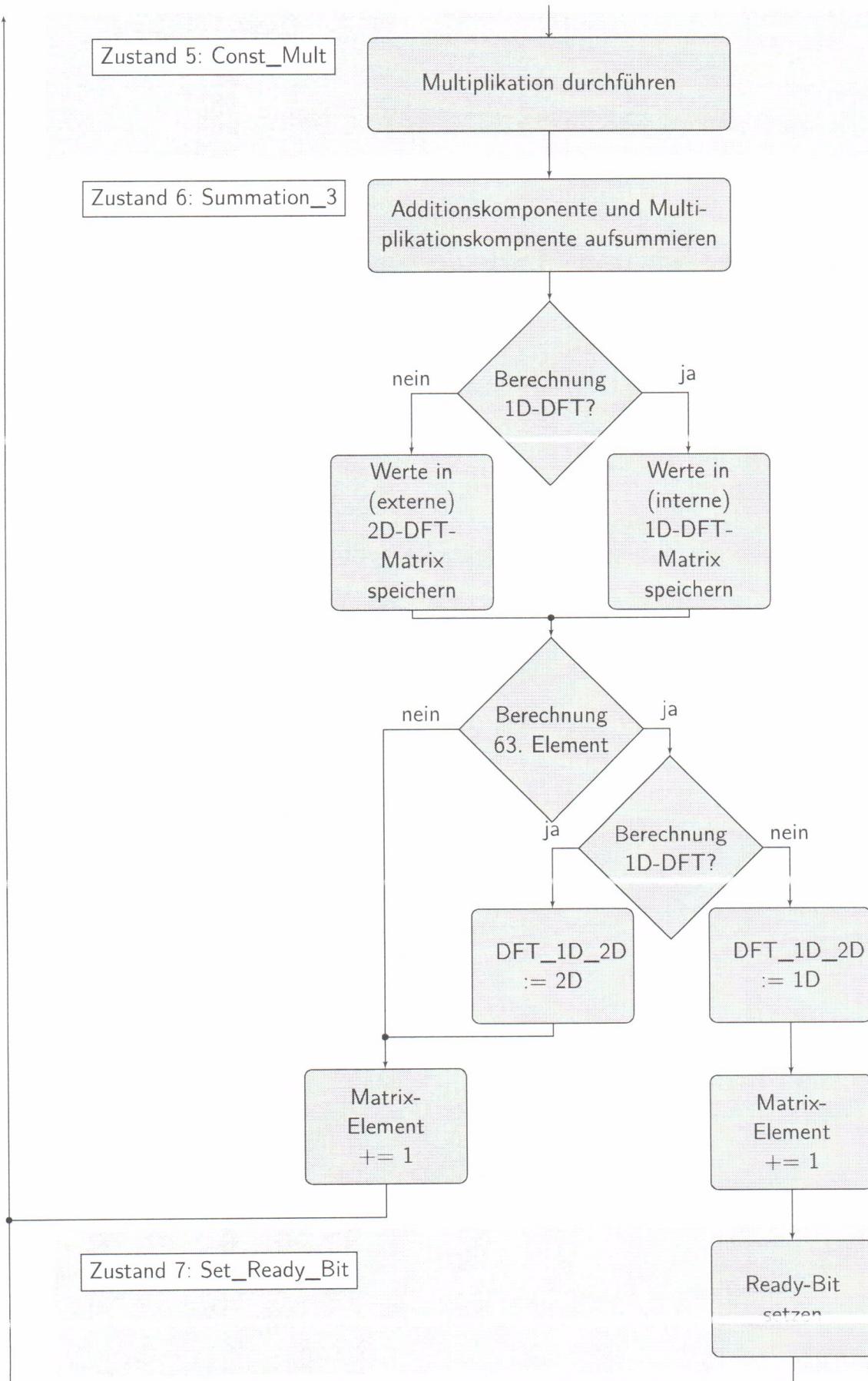


Abbildung 4.7: Automatengraf

4.9 UML-Diagramm







4.10 Projekt- und Programmstruktur

Konstanten

Datentypen

readfile (read_input_matrix)

writefile (write_results)

resize-Funktion

4.11 Bibliotheken und Hardwarebeschreibungssprache

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library STD;
use STD.TEXTIO.ALL;
use ieee.std_logic_textio.all;
VHDL 2008 kann auch Kommazahlen darstellen ( signed fixed : sfixed(2 downto -10) )
```