Philipp W. Besslich · Tian Lu

Diskrete Orthogonal-transformationen

Algorithmen und Flußgraphen für die Signalverarbeitung

Mit 91 Abbildungen

Springer-Verlag Berlin Heidelberg GmbH

Prof. Dr.-Ing. Philipp W. Besslich
Institut für Theoretische Elektrotechnik
und Digitale Systeme
FB1/Universität Bremen
2800 Bremen

Dipl.-Ing. Tian Lu
Steegerstraße 11a
1000 Berlin 65

CIP-Titelaufnahme der Deutschen Bibliothek
Besslich, Philipp:
Diskrete Orthogonaltransformationen : Algorithmen und Flußgraphen für die Signalverarbeitung /
Philipp W. Besslich ; Tian Lu.
Berlin ; Heidelberg ; New York ; London ; Paris ; Tokyo ; Hong Kong : Springer, 1990

ISBN 978-3-540-52151-8 ISBN 978-3-642-48933-4 (eBook)
DOI 10.1007/978-3-642-48933-4

NE: Tian, Lu

Ursprünglich erschienen bei Springer-Verlag Berlin Heidelberg New York 1990.

2068/3020 543210 - Gedruckt auf säurefreiem Papier

Vorwort

Seit Bekanntwerden "schneller" Algorithmen haben orthogonale Transformationen in der digitalen Signalverarbeitung zunehmend an Bedeutung gewonnen. Ausgehend vom "klassischen" Cooley-Tukey-Algorithmus der diskreten Fourier-Transformation wurde eine Vielzahl anderer Algorithmen mit dem Ziel entwickelt, den Rechenaufwand weiter zu reduzieren. So entstanden z.B. der Primfaktor-Algorithmus, der Winograd-Algorithmus und der Split-Radix-Algorithmus der DFT. Außerdem wurden schnelle Algorithmen für andere Transformationen, wie die Walsh-, Haar-, Cosinus-Transformation etc. bekannt. Viele dieser Transformationsalgorithmen können vom Rechner stufenweise, d.h. in Iterationen abgearbeitet werden, so daß i.a. eine speichersparende "in-place"-Verarbeitung möglich ist.

Wegen der rasanten Entwicklung auf dem Gebiet der digitalen Bildverarbeitung hat das Interesse an zweidimensionalen Orthogonaltransformationen in letzter Zeit stark zugenommen. Infolgedessen entstanden zahlreiche Algorithmen für 2D-Transformationen. Häufig wird hierbei von der Separierbarkeit in Zeilen- und Spaltentransformationen Gebrauch gemacht. Es hat sich jedoch gezeigt, daß anders geartete Algorithmen (z.B. die Berechnung aus Koeffizienten niedrigerer Ordnung) vorteilhafter sein können.

Vor dem Hintergrund der Verfügbarkeit von integrierten Schaltungen zur Durchführung orthogonaler Transformationen gewinnt die Signalverarbeitung im Transformationsbereich eine neue Dimension. Die Möglichkeit, spezielle Transformationsalgorithmen als (semi-) kundenspezifische integrierte Schaltungen zu realisieren, läßt die Kenntnis bestehender Algorithmen sowie der ihnen zugrunde liegenden Prinzipien besonders aktuell erscheinen.

Mit diesem Buch werden mehrere Ziele verfolgt: Nach einer Übersicht über die wichtigsten Orthogonaltransformationen (Kapitel 2) werden einige grundsätz-

liche Probleme der diskreten "schnellen" Transformations-Algorithmen behandelt (Kapitel 3). Dazu zählen u.a.: Komplexität, Berechnungs-Strategien (z.B. direkte und indirekte Berechnung der Transformationskoeffizienten), Anwendung der Restklassenarithmetik, Faktorisierung der Transformationsmatrizen sowie der Zusammenhang zwischen der Iterationsfolge und der Reihenfolge von Objektdaten und Koeffizienten. Für zweidimensionale Transformationen treten weitere Fragestellungen hinzu, wie Separierbarkeit, Zurückführung auf eine eindimensionale Transformation, hierarchische Berechnung, etc. (Kapitel 5).

Basierend auf diesen allgemeinen Überlegungen werden die wichtigsten Algorithmen für ein- und zweidimensionale Orthogonaltransformationen (Kapitel 4 und 6) vorgestellt. Dabei wird auf eine vereinheitlichte Terminologie Wert gelegt, sowohl bei den Herleitungen wie bei den zugehörigen Signalflußgraphen.

Anwendung finden die hier vorgestellten Transformationsalgorithmen in Programmen für Allzweckrechner, Mikro- oder Signalprozessoren oder für spezielle Transformations-Hardware. Im ersten Fall werden die Programme i.a. in einer Hochsprache geschrieben, während Mikro- und Signalprozessoren häufig in ihren Assembler-Sprachen programmiert werden. Insbesondere bei der Transformation größerer Bildmatrizen können jedoch die Verarbeitungszeiten beträchtlich sein. Deshalb ist man bemüht, hierfür spezielle parallelverarbeitende Architekturen zu entwickeln. Abschließend (Kapitel 7) werden Hinweise auf die Implementierung der Algorithmen in Software und Hardware gegeben.

Dieses Buch wendet sich in erster Linie an Ingenieure und Studenten der Informationstechnik, aber auch an Informatiker und Wissenschaftler anderer Disziplinen, die vor der Aufgabe stehen, einen geeigneten Algorithmus einer ein- oder zweidimensionalen Transformation in Software oder Hardware zu implementieren. Wegen der Erörterung einiger allgemeiner Gesichtspunkte mag das Buch auch für diejenigen von Nutzen sein, die sich mit der Entwicklung neuer Algorithmen befassen.

Herrn Prof. Dr. P. Noll, Institut für Fernmeldetechnik der TU Berlin, danken wir vielmals für seine Föderung bei der Erstellung des Manuskripts. Wir danken gleichfalls Herrn Prof. Dr. A. Wasiljeff, Institut für Telekommunika-

tionstechnik der Universität Bremen für fruchtbare Diskussionen sowie Frau Ch. Stinner für die graphische Gestaltung. Ein besonderer Dank gebührt Herrn Dr. James W. Cooley, IBM T. J. Watson Research Center, Yorktown Heights, NY, der es trotz drängender Termine ermöglichte, unser Manuskript durchzusehen und Anregungen zu einigen Verbesserungen zu geben.

Bremen, Frühjahr 1990 — Philipp W. Besslich

Berlin — Tian Lu

Inhaltsverzeichnis

Abkürzungen und Formelzeichen XII

1 Einführung 1
1.1 Anwendungsbereiche von Orthogonaltransformationen in der Signalverarbeitung 1
1.2 Diskrete orthogonale Transformationen 5
1.3 Darstellungsformen und Komplexität der Transformationsalgorithmen 8
1.4 Organisation des Buches 10

2 Die wichtigsten diskreten Orthogonaltransformationen 12
2.1 Fourier-Transformation 12
2.2 Hartley-Transformation 17
2.3 Cosinus- und Sinus-Transformation 19
2.4 Walsh-Hadamard-Transformation 21
2.5 Haar-Transformation 34
2.6 Slant-Transformation 37

3 Strategien, Berechnungs- und Darstellungsformen 43
3.1 Berechnungsstrategien 43
3.1.1 Direkte Berechnung der Transformationskoeffizienten 43
3.1.2 Berechnung aus Koeffizienten von Teilfolgen 43
3.1.3 Berechnung aus den Koeffizienten einer anderen Transformation 44
3.2 Anwendung der Restklassenarithmetik 45
3.3 Faktorisierung der Transformationsmatrix 50

3.4 Indizierung der Daten im Objekt- und im Transformationsbereich ... 57
3.4.1 Aufteilung in Module ... 57
3.4.2 Darstellung durch Signalflußgraphen ... 60
3.4.3 Beziehungen zwischen Iterationenfolge, Objektdaten- und Koeffizientenreihenfolge ... 64

4 Algorithmen eindimensionaler Transformationen ... 74
4.1 Algorithmen für die diskrete Walsh-Transformation ... 74
4.1.1 Algorithmen für die Paley-, Hadamard- und Walsh-Ordnung ... 74
4.1.2 Berechnung der WHT aus Koeffizienten von Teilfolgen ... 77
4.2 Algorithmen für die diskrete Fourier-Transformation ... 78
4.2.1 Cooley-Tukey- und Sande-Tukey-Algorithmus ... 80
4.2.2 Radix-4- und Split-Radix-Algorithmus für die FFT ... 90
4.2.3 Primfaktor-Algorithmus für die DFT ... 97
4.2.4 Winograd-Algorithmus ... 103
4.2.5 Berechnung der DFT aus Walsh-Koeffizienten ... 107
4.2.6 Berechnung der DFT aus Koeffizienten von Teilfolgen ... 113
4.2.7 FFT-Algorithmus für reelle, symmetrische oder antisymmetrische Datensequenzen ... 120
4.3 Algorithmen für die diskrete Hartley-Transformation ... 130
4.3.1 DHYT-Algorithmus mit Zeitdezimierung ... 131
4.3.2 DHYT-Algorithmus mit Frequenzdezimierung ... 134
4.3.3 Primfaktor-Algorithmus für die DHYT ... 136
4.3.4 Split-Radix-Algorithmus für die DHYT ... 138
4.3.5 Berechnung der DHYT aus Koeffizienten von Teilfolgen ... 141
4.3.6 DHYT-Berechnung aus Walsh-Koeffizienten ... 146
4.3.7 Konversion zwischen DFT- und DHYT-Koeffizienten ... 151
4.4 Algorithmen für die diskrete Cosinus-Transformation ... 153
4.4.1 DCT-Berechnung mittels 2N-Punkte-DFT ... 154
4.4.2 DCT-Berechnung mittels N-Punkte-DFT ... 159
4.4.3 DCT-Berechnung aus DHYT-Koeffizienten ... 165
4.4.4 DCT-Algorithmus mit Matrixfaktorisierung ... 167
4.4.5 Algorithmus zur direkten DCT-Berechnung ... 174
4.4.6 DCT-Berechnung aus Koeffizienten von Teilfolgen ... 178
4.4.7 Algorithmus für eine modifizierte DCT ... 182

4.5 Algorithmen für die diskrete Sinus-Transformation 187
4.5.1 DST-Algorithmus mit Matrixfaktorisierung 187
4.5.2 DST-Berechnung über die DCT 191
4.6 Algorithmen für die diskrete Haar-Transformation 193
4.6.1 HT-Algorithmus ohne "in-place"-Eigenschaft 194
4.6.2 Algorithmen zur "in-place"-Berechnung der HT 196
4.7 Algorithmen für die Slant-Transformation 198
4.7.1 SLT-Algorithmus mit Faktorisierung der Transformationsmatrix 198
4.7.2 SLT-Algorithmus unter Verwendung der Walsh-Transformation . 198
4.7.3 SLT-Algorithmus unter Verwendung der Haar-Transformation .. 203

5 Berechnungsstrategien zweidimensionaler Transformationen 204
5.1 Berechnung der 2D-Koeffizienten durch eine 1D-Transformation (und umgekehrt) 208
5.2 Separierung in Zeilen- und Spaltentransformation 211
5.3 Direkte Berechnung der 2D-Koeffizienten aus der 2D-Datenmenge ... 215
5.4 Berechnung aus 2D-Koeffizienten niedrigerer Ordnung 221

6 Algorithmen zweidimensionaler Transformationen 225
6.1 Algorithmen für die 2D-Walsh-Transformation 225
6.2 Algorithmen für die 2D-Fourier-Transformation 231
6.2.1 Zeile/Spalte-Algorithmus der DFT 233
6.2.2 Vektor-Radix-Algorithmus der DFT 235
6.2.3 Direkter 2D-FFT-Algorithmus 241
6.2.4 Berechnung der 2D-FFT aus Koeffizienten niedrigerer Ordnung 250
6.2.5 Vereinheitlichte Behandlung der zweidimensionalen DFT-Algorithmen 256
6.2.6 DFT-Algorithmen für reelle 2D-Datenarrays 266
6.3 Algorithmen für die diskrete 2D-Hartley-Transformation 270
6.3.1 Zeile/Spalte-Algorithmus der DHYT 271
6.3.2 Vektor-Radix-Algorithmus der DHYT 273
6.4 Algorithmen für die diskrete 2D-Cosinus-Transformation 277

6.4.1 DCT-Algorithmus zur direkten Anwendung auf die 2D-Datenmenge 279
6.4.2 DCT-Berechnung aus Koeffizienten niedrigerer Ordnung 289

7 Implementierung der Algorithmen 293
7.1 Software-Implementierung 293
7.2 Hardware-Realisierung 296

Literaturverzeichnis 301

Sachverzeichnis 309

Abkürzungen und Formelzeichen

Walsh-Transformation:

$wal(k,t)$	kontinuierlich definierte Walsh-Funktion
$Wal(k,n)$	diskrete Walsh-Funktion
$wal_{w}(k,t)$	nach Walsh geordnete wal(k,t)
$Wal_{W}(k,n)$	nach Walsh geordnete Wal(k,n)
WHT	Walsh-Hadamard-Transformation
$(WHT)_{W}$	nach Walsh geordnete WHT
$X_{W}(k)$	$(WHT)_{W}$-Koeffizienten von x(n)
$[H_{W}]$	$(WHT)_{W}$-Matrix
$h_{W}(k,n)$	Elemente von $[H_{W}(L)]$
$wal_{h}(k,t)$	nach Hadamard geordnete wal(k,t)
$Wal_{H}(k,n)$	nach Hadamard geordnete Wal(k,n)
$(WHT)_{H}$	nach Hadamard geordnete WHT
$X_{WH}(k)$	$(WHT)_{H}$-Koeffizienten von x(n)
$[H_{H}]$	$(WHT)_{H}$-Matrix
$h_{H}(k,n)$	Elemente von $[H_{H}(L)]$
$wal_{p}(k,t)$	nach Paley geordnete wal(k,t)
$Wal_{P}(k,n)$	nach Paley geordnete Wal(k,n)
$(WHT)_{P}$	nach Paley geordnete WHT
$X_{WP}(k)$	$(WHT)_{P}$-Koeffizienten von x(n)
$[H_{P}]$	$(WHT)_{P}$-Matrix
$h_{P}(k,n)$	Elemente von $[H_{P}(L)]$
$cal(k,t)$	kontinuierlich definierte Cal-Funktion
$Cal(k,n)$	diskrete Cal-Funktion
$sal(k,t)$	kontinuierlich definierte Sal-Funktion
$Sal(k,n)$	diskrete Sal-Funktion
$wal_{cs}(k,t)$	wal(k,t) in Cal-Sal-Ordnung
$Wal_{CS}(k,n)$	Wal(k,n) in Cal-Sal-Ordnung

$(WHT)_{CS}$	nach Cal-Sal geordnete WHT
$X_{CS}(k)$	$(WHT)_{CS}$-Koeffizienten von x(n)
$[H_{CS}]$	$(WHT)_{CS}$-Matrix
$h_{CS}(k,n)$	Elemente von $[H_{CS}(L)]$
FWHT	schnelle Walsh-Hadamard-Transformation

Fourier-Transformation:

DFT	diskrete Fourier-Transformation
IDFT	inverse DFT
FFT	schnelle Fourier-Transformation
$X_f(k)$	Fourier-Koeffizienten von x(n)
$[W_N^E]$	Fourier-Transformationsmatrix

Hartley-Transformation:

DHYT	diskrete Hartley-Transformation
IDHYT	inverse DHYT
$X_{HY}(k)$	Hartley-Koeffizienten von x(n)
[HY]	Hartley-Transformationsmatrix
FHYT	schnelle Hartley-Transformation

Cosinus-Transformation:

DCT	diskrete Cosinus-Transformation
IDCT	inverse DCT
$X_c(k)$	Cosinus-Koeffizienten von x(n)
[C]	Cosinus-Transformationsmatrix
FCT	schnelle Cosinus-Transformation

Sinus-Transformation:

DST	diskrete Sinus-Transformaton
IDST	inverse DST
$X_s(k)$	Sinus-Koeffizienten von x(n)
[S]	Sinus-Transformationsmatrix
FST	schnelle Sinus-Transformation

Haar-Transformation:

har(k,t)	kontinuierlich definierte Haar-Funktion

HT	Haar-Transformation
$X_{HA}(k)$	Haar-Koeffizienten von x(n)
[HAR]	Haar-Transformationsmatrix
FHT	schnelle Haar-Transformation

Slant-Transformation:

sla(k,t)	kontinuierlich definierte Slant-Funktion
SLT	Slant-Transformation
$X_{SL}(k)$	Slant-Koeffizienten von x(n)
[SL]	Slant-Transformationsmatrix

Anzahl von Operationen:

A_r	Anzahl der zu einer Berechnung benötigten reellen Additionen
A_c	Anzahl der zu einer Berechnung benötigten komplexen Additionen
M_r	Anzahl der benötigten reellen Multiplikationen
M_c	Anzahl der benötigten komplexen Multiplikationen
C_r	Gesamtzahl der benötigten reellen Operationen
C_c	Gesamtzahl der benötigten komplexen Operationen

Allgemeine Abkürzungen:

BEO	Bitaustauschoperation (Bit Exchange Operation)
BRO	Bitumkehroperation (Bit Reversal Operation)
CRT	Chinesisches Rest-Theorem
GGT	größer gemeinsamer Teiler
PFA	Primfaktor-Algorithmus
SFG	Signalflußgraph
SR	Split-Radix
VR	Vektor-Radix
WFTA	Winograd-Fourier-Transformationsalgorithmus

Formelzeichen:

$[A]^T$	Transponierte der Matrix [A]
$[\overline{A}]$	gespiegelte Matrix [A]
$\underline{B}$	Vektor B
f^*	konjugiert komplexer Ausdruck zu f
f(x)*g(x)	lineare Faltung

$[I_N]=[I(r)]$	Einheitsmatrix der Ordnung r mit $N=2^r$
k	Laufindex im Frequenzbereich
<k>	BRO von k
X(k)	Datensatz im Transformationsbereich (Koeffizienten)
L	Laufindex der Matrixordnung (L=1,2,...,r)
n	Laufindex im Objektbereich
x(n)	Datensatz im Objektbereich
N	Anzahl der Datenwerte bzw. Transformationskoeffizienten
$\lceil z \rceil$	kleinste ganze Zahl größer oder gleich z
$<z>_N$	Reduktion von z modulo N (z mod N)
<——>	Korrespondenzzeichen (z.B. x(n) <——> X(k))
$\oplus$	Operationszeichen der Modulo-2-Addition
$\otimes$	Operationszeichen für das Kronecker-Produkt

Die vorstehende Liste von Abkürzungen und Formelzeichen enthält die wichtigsten Bezeichnungen, die einheitlich im gesamten Buch benutzt werden. Darüberhinaus kommen zahlreiche lokal definierte Zeichen zur Anwendung.

1 Einführung

1.1 Anwendungsbereiche von Orthogonaltransformationen in der Signalverarbeitung

Gegenstand dieses Buches sind Algorithmen diskreter Transformationen für die digitale Signalverarbeitung. Im Vordergrund des Interesses stehen dabei die orthogonalen (bzw. unitären) Transformationen. Ein- und zweidimensionale diskrete Orthogonaltransformationen sind als Hilfsmittel in der Signalverarbeitung weit verbreitet. Da sie auf modernen Rechnern effizient eingesetzt werden können, sind sie auch aus vielen anderen Bereichen der Natur- und Ingenieurwissenschaften nicht mehr fortzudenken. In der digitalen Bildverarbeitung spielen zweidimensionale Transformationen eine wichtige Rolle. Eine Sonderstellung nehmen auch hier die umkehrbar eindeutigen Transformationen ein, da durch sie keine Informationsreduktion eintritt, sondern lediglich eine äquivalente Darstellung gewonnen wird.

Bevor wir auf die Anwendungsmöglichkeiten orthogonaler Transformationen eingehen, muß eine geeignete Terminologie eingeführt werden. Durch die Transformationen werden Daten eines *Objektbereichs* in einen *Transformationsbereich* überführt. Der Objektbereich kann verschiedener Natur sein: Er kann nicht nur ein- oder mehrdimensional sein, sondern auch von verschiedener *physikalischer* Dimension. Eindimensionale Transformationen werden in der Informationstechnik häufig auf Zeitfunktionen angewandt. Falls es sich um eine Fourier-Transformation handelt, hat die Variable des Transformationsbereichs die Dimension einer Frequenz. Selbstverständlich braucht der Objektbereich einer eindimensonalen Fourier-Transformation nicht der "Zeitbereich" zu sein. Dann ist jedoch die Anwendung des Begriffs *Frequenz* mit der Dimension s^{-1} im Transformationsbereich nicht gegeben. Ist der Objektbereich etwa der Ortsbereich, so ist die Dimension des Transformationsbereichs m^{-1}, z.B. Li-

nienpaare pro mm. In diesem Fall spricht man häufig von *Ortsfrequenz*. Es ist weiterhin zu beachten, daß der Begriff der (zeitbezogenen) Frequenz nur für trigonometrische Basisfunktionen erklärt ist. Er ist deshalb auf die Walsh-, Haar- und andere nicht-trigonometrische Basisfunktionen nicht anwendbar. Für diese Art der (Basis-) Funktionen wurde deshalb der Begriff *Sequenz* vorgeschlagen [1.1]. Diese ist als die halbe Anzahl der Nulldurchgänge pro Einheit des Objektbereichs definiert. Entsprechendes gilt gleichermaßen für mehrdimensionale Transformationen. Eine wichtige Anwendung betrifft die zweidimensonalen Transformationen in der Bildverarbeitung [1.2]. Hier ist der Objektbereich ein zweidimensionaler Ortsbereich, der Transformationsbereich ist der ebenfalls zweidimensionale Ortsfrequenzbereich. Den manchmal verwendeten Begriff "Bildbereich" vermeiden wir wegen der damit verbundenen Möglichkeiten der Konfusion.

In diesem Buch behandeln wir ausschließlich diskrete Transformationen. Es wird also vorausgesetzt, daß ein ein- oder mehrdimensionales Signal bereits in abgetasteter Form vorliegt. Zwar sind die meisten Signale, auf die die hier behandelten Transformationen angewandt werden, analoger Natur. Für die rechnerische Durchführung der Transformation wird jedoch eine digitale Variante benötigt. Zu ihrer Erzeugung ist eine Abtastung im Objektbereich und eine Quantisierung der Funktionswerte erforderlich. Auf die hierdurch entstehenden Quantisierungsfehler sowie auf Fehler durch Verletzung des Abtasttheorems (Aliasing) und durch die nicht-ideale (technische) Abtastung wird hier nicht eingegangen. Wir setzen also die Existenz von N quantisierten Abtastwerten (oder Stützstellen) für eine eindimensionale (1D-) Transformation, bzw. von $N_1 \times N_2$ Abtastwerten für eine 2D-Transformation voraus. Durch die Transformation werden im allgemeinen N bzw. $N_1 \times N_2$ Transformationskoeffizienten erzeugt.

Der Anwendungsbereich orthogonaler Transformationen ist weit gespannt. Man überblickt die vielschichtigen Anwendungsmöglichkeiten vielleicht am besten, indem man sich die verschiedenen Interpretationsmöglichkeiten linearer und speziell orthogonaler Transformationen vor Augen hält. Eine dieser Interpretationen kann man direkt aus ihrer Definition ablesen (vgl. (1.1)). Diese allgemeinste Art der Darstellung läuft auf eine Reihenentwicklung des Signals nach orthogonalen Basisfunktionen hinaus. Dabei geben die Transformationskoeffizienten an, mit welchem Anteil die zugehörigen Basisfunktionen im

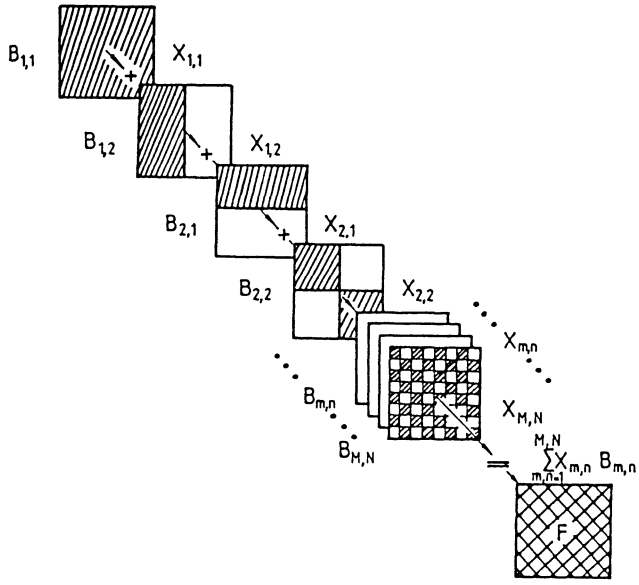


Bild 1.1. Schematische Darstellung einer zweidimensionalen Funktion F (Bild) durch eine Summe von zweidimensionalen Basisfunktionen $B_{m,n}$ der Gewichte $X_{m,n}$

Signal enthalten sind. Ein Beispiel der Zerlegung eines 2D-Signals in seine Basisbilder zeigt Bild 1.1. Man erkennt, daß die Koeffizienten ein Maß für die Ähnlichkeit des Signals mit den jeweiligen Basisbildern darstellen. Hieraus folgt z.B. die Möglichkeit, Transformationskoeffizienten für die Klassifikation von Mustern heranzuziehen, z.B. in Form der sogenannten Fourier-Deskriptoren.

Eine andere Betrachtungsweise ist die weit verbreitete Darstellung der Transformationskoeffizienten als Frequenzspektrum. Sie ist eng verknüpft mit der Theorie der Frequenzfilter (Tiefpässe, Hochpässe, usw.) [1.3], [1.4]. Streng genommen ist diese Betrachtungsweise nur auf die Fourier-Transformation von (eindimensionalen) Zeitfunktionen anwendbar. Jedoch kann man sie durch geeignete Definitionen auch auf andere Fälle erweitern, z.B. auf Ortsfrequenzen von (zweidimensionalen) Bildsignalen [1.5]. Eine analoge Erweiterung auf nicht-trigonometrische Transformationen ist das Sequenz-Spektrum [1.6]. Diesbezügliche Anwendungen der Orthogonaltransformationen liegen auf

dem Gebiet des ein- oder mehrdimensionalen Filterentwurfs, z.B. von zweidimensionalen Digitalfiltern für die Bildverarbeitung.

Eine weitere Variante in der Betrachtungsweise orthogonaler Transformationen ist die Koordinatentransformation. Die Objektdaten können als N-dimensionaler Vektor interpretiert werden. Man beachte, daß die "Dimensionalität" dieses Vektors gleich der Anzahl der Abtastwerte ist und nicht mit der eigentlichen physikalischen Dimension des Objektbereichs verwechselt werden darf. Die Anwendung einer orthogonalen Transformation auf diesen Vektor entspricht dann einer Rotation des Koordinatensystems im N-dimensionalen Raum. Die Transformationskoeffizienten stellen somit die Datenwerte im neuen (rotierten) Koordinatensystem dar. Eine derartige Betrachtungsweise ist z.B. bei der Anwendung zur Transformationskodierung von Bildern sinnvoll [1.7]. Hier erwartet man von der Transformation eine Umverteilung der Varianzen der ursprünglichen Daten (Grauwerte der Bildelemente) auf die transformierten Daten im rotierten Koordinatensystem (Transformationskoeffizienten) derart, daß die Varianzen (Signalleistung) auf möglichst wenige Koeffizienten konzentriert werden. Eine vereinfachte Darstellung dieses Sachverhalts zeigt Bild 1.2. Dabei handelt es sich um die sog. Korrelationsellipse eines Grauwert-Testbildes. Die Grauwerte x_1 und x_2 benachbarter Bildpunkte sind als zweidimensionale Vektoren in das zweidimensionale Koordinatensystem (x_1, x_2) eingetragen. Sichtbare Häufungen in der Region $x_1 \approx x_2$ verdeutlichen die Korrelation benachbarter Bildpunkte. Die Varianz der Grauwerte ist offensichtlich ziemlich gleichmäßig auf die Koordinanten x_1 und x_2 verteilt. Durch Rotation des Koordinatensystems gelingt es, die Abhängigkeit der Daten von x_2 zu verkleinern und die von x_1 entsprechend zu vergrößern. Mit anderen Worten: Die Koordinatentransformation bewirkt eine Umverteilung der Varianzen (bei konstanter Summe).

Bei der Anwendung der Orthogonaltransformationen zur Transformationskodierung ist es das Ziel der Transformationsoperation, die zwischen benachbarten Abtastwerten bestehenden Korrelationen möglichst aufzuheben. Es läßt sich zeigen, daß man bei den sogenannten optimalen Transformationen (Karhunen-Loève-Transformation und Singulärwertzerlegung) eine völlige Dekorrelation der Koeffizienten erhält. Allerdings sind die Basisfunktionen hierbei vom statistischen Signalmodell (Karhunen-Loève-Transformation) bzw. vom Bildinhalt selbst (Singulärwertzerlegung) abhängig, d.h. sie müssen für jedes Mo-

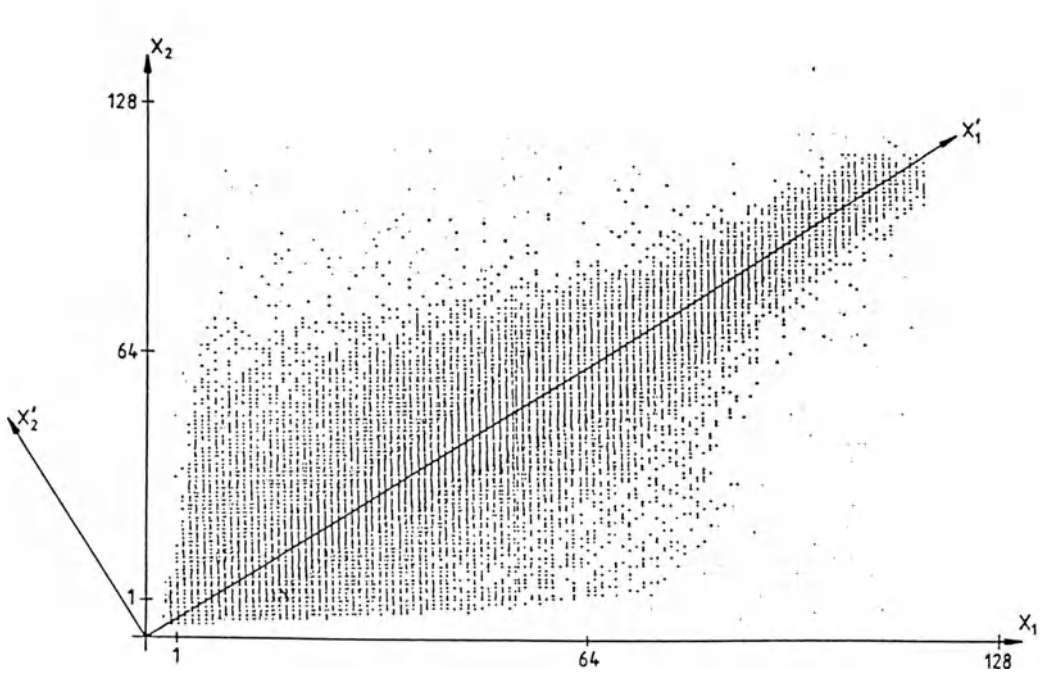


Bild 1.2. Umverteilung von Varianzen durch Rotation des Koordinatensystems, nach [1.7]

dell bzw. jedes Bild erst bestimmt werden. Alle Orthogonaltransformationen mit festen Basisfunktionen ergeben nur suboptimale Dekorrelationsergebnisse. Da es naturgemäß für die optimalen Transformationen keine schnellen Algorithmen gibt, sind sie nicht Gegenstand dieses Buches.

1.2 Diskrete orthogonale Transformationen

Zunächst betrachten wir die Eigenschaften und die Darstellungsformen der orthogonalen (bzw. der unitären) Transformationen. Es sei

$$X_k = \sum_{n=0}^{N-1} a_{kn} x_n + b_n, \qquad k = 0,1,2,\ldots,N-1,$$

wobei die x_n Elemente einer Menge von Daten der Mächtigkeit N sind. Die Gewichte a_{kn} können als Elemente einer Matrix $[A] = [a_{kn}]$ gedacht werden. Falls $\det|A| \neq 0$ gilt, ist die Transformation regulär und invertierbar

$$x_n = \sum_{k=0}^{N-1} a_{kn}^{-1} X_k + c_n \quad \text{mit} \quad c_n = -\sum_{k=0}^{N-1} a_{kn}^{-1} b_k .$$

In Matrixnotation schreibt man für diese eindeutig umkehrbare Transformation

$$\underline{X} = [A] \cdot \underline{x} + \underline{b} \qquad \text{bzw.} \quad \underline{\hat{x}} = [A]^{-1} \underline{X} + \underline{c},$$

wobei die Vektoren die Ordnung N und die Matrizen die Ordnung NxN haben. Ihre Elemente sind beliebige komplexe Zahlen. Wir betrachten den Sonderfall b=c=0 (affine Transformationen). Wenn die euklidischen Abstände in der komplexen Ebene invariant sind, nennt man die Transformationen unitär. Bei Einschränkung auf reelle Werte heißen sie orthogonal.

Wenn das Skalarprodukt zweier Vektoren $\underline{a}$ und $\underline{b}$ Null ist, d.h. $\underline{a} \cdot \underline{b} = 0$, nennt man die Vektoren zueinander orthogonal. Nun betrachten wir eine Matrix [A] der Ordnung cxr. Falls $[A][A]^T=[D]$ gilt, sind die Spaltenvektoren von [A] paarweise orthogonal. Dabei ist [D] eine Diagonalmatrix. Ist [D]=[I], wobei [I] die Einheitsmatrix ist, dann nennt man die Vektoren orthonormal. Im Falle einer quadratischen Matrix (c=r) heißt diese orthogonal, wenn $[A][A]^T= [I]$, d.h. es gilt $[A]^T= [A]^{-1}$. Eine orthogonale Matrix ist also notwendig nichtsingulär, d.h. ihre Zeilen- bzw. Spaltenvektoren sind linear unabhängig.

Für eine unitäre Matrix gilt entsprechend $[A^*]^T= [A]^{-1}$. Hierbei bedeutet der Stern, daß alle Elemente konjugiert komplex sind. Im Sonderfall der symmetrischen orthogonalen Matrix ist $a_{kn} = a_{nk}$ und $[A] = [A]^{-1}$.

Weitere Hinweise bezüglich der Anwendung der Matrizenrechnung für die Transformationsalgorithmen finden sich im Abschnitt 3.3.

Bei den in der digitalen Signalverarbeitung gebräuchlichen Transformationen handelt es sich i.allg. um quadratische Systeme, d.h. solche, bei denen die Zahl der Koeffizienten gleich der Zahl der Datenelemente im Objektbereich ist. Die Transformationsmatrix [G] enthält dann die konjugiert komplexen (oder reellen) Werte einer unitären (oder orthogonalen) Approximationsfunktion $g_k(n)$, $n = 0,1,2,\ldots,N-1$:

$$[G] = \begin{bmatrix} g_0^*(0) & \cdots & g_0^*(N-1) \\ & \ddots & \\ & g_i^*(j) & \\ & & \ddots \\ g_{N-1}^*(1) & \cdots & g_{N-1}^*(N-1) \end{bmatrix}.$$

Die Menge der Funktionen $g_k(n)$ stellen den Transformationskern dar. Für unitäre bzw. orthogonale Kernsysteme $\{g_k(n)\}$ ist dabei

$$g_i(n) \cdot g_j^*(n) = \begin{cases} c^2 & \text{für } j=i \\ 0 & \text{sonst.} \end{cases}$$

Im Fall von orthogonalen Kernen sind die $g_k(n)$ reell und der Stern entfällt. Wenn $c^2 = 1$ ist, spricht man von orthonormalen Systemen. Die Koeffizienten X(k) der diskreten unitären (oder orthogonalen) Transformation sind wie folgt definiert:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot g_k^*(n) \quad \text{bzw.} \quad \underline{X} = [G] \cdot \underline{x}. \tag{1.1}$$

Bei Anwendung der Besselschen Ungleichung auf das quadratische System geht diese über in die Parsevalsche Gleichung (oder Vollständigkeitsrelation)

$$\sum_{k=0}^{N-1} X^2(k) = \sum_{n=0}^{N-1} x(n) \cdot x^*(n).$$

Nur im Fall der Vollständigkeit des approximierenden Funktionensystems (Transformationskerns) $g_k(n)$ erfolgt eine umkehrbar eindeutige Zuordnung der Vektoren $\underline{x}$ und $\underline{X}$.

Das hat u.a. zur Folge, daß es keine Vektoren $\underline{x}$ und $\underline{X}$ geben kann, die durch das Transformationspaar

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X(k) \cdot g_k(n) \quad \text{und} \quad X(k) = \sum_{n=0}^{N-1} x(n) \cdot g_k^*(n) \tag{1.2}$$

nicht darstellbar wären.

Beispiel eines zwar orthogonalen aber nicht vollständigen Kerns ist das System $\sin(\pi kn)/(\pi kn)$. Mit ihm können nur bandbegrenzte Funktionen interpoliert werden.

Der normalisierende Faktor N^{-1} in (1.2) wird in der Literatur in unterschiedlicher Weise der Hin- oder der Rücktransformation zugeordnet. Manche Autoren teilen den Faktor auch gleichmäßig auf beide Transformationen auf, was von Standpunkt des Rechenaufwandes jedoch unzweckmäßig ist. Da die inverse Transformation weniger häufig benutzt wird, teilen wir i.allg. dieser den Normalisierungsfaktor zu. Bei der Matrizenschreibweise (vgl. (1.3)) verzichten wir gelegentlich auf den Normalisierungsfaktor, d.h. wir setzen voraus, daß dieser in die inverse Transformationsmatrix absorbiert sei.

1.3 Darstellungsformen und Komplexität der Transformationsalgorithmen

Die Transformationsalgorithmen, d.h. die Rechenvorschriften zur Ermittlung von N Transformationskoeffizienten aus den N Werten der Objektsequenz (Stützstellen) sowie die hierzu inverse Operation, können in verschiedener Weise dargestellt werden:

(1) Es liegt nahe, die Summenausdrücke (1.2) direkt zu benutzen. Bei der Berechnung sind N Summen zu bilden, die im allgemeinen aus je N Produkten bestehen. Wie sich zeigen wird, ist der hierzu benötigte Rechenaufwand beträchtlich und kann u.U. erheblich reduziert werden (vgl. Abschnitt 3.3).

(2) Eine bequemere, aber doch gleichwertige Darstellung erhält man bei Anwendung der Matrixnotation. Die Summen von Produkten, die in (1.2) enthalten sind, können natürlich auch wie folgt geschrieben werden:

$$\underline{X} = [G]\cdot\underline{x} \quad \text{bzw.} \quad \underline{x} = N^{-1}\cdot[G^*]^T\cdot\underline{X}. \tag{1.3}$$

Hierbei ist [G] die Transformationsmatrix der Ordnung NxN und $\underline{x}$ bzw. $\underline{X}$ sind die Spaltenvektoren des Objektdatensatzes und der Transformationskoeffizienten.

Die vereinfachte Schreibweise darf nicht darüber hinwegtäuschen, daß gegenüber der Summendarstellung keinerlei Einsparungen im Rechenaufwand erhalten werden. Hingegen bietet die Matrixdarstellung (1.3) den Ansatzpunkt für eine Faktorisierung (vgl. Abschnitt 3.3), die erhebliche Einsparungen des Rechenaufwandes zur Folge hat.

(3) Als Hilfsmittel zur Programmierung von Algorithmen finden häufig Signalflußgraphen Anwendung. Ein Signalflußgraph ist ein gerichteter Graph, dessen Knoten Speicherplätze darstellen, und dessen Pfeile die Kombination der Daten verdeutlichen. Die Anwendung von Signalflußgraphen erhält eine besondere Bedeutung, wenn es gelingt, die Matrizen [G] so zu faktorisieren, daß die Koeffizienten schrittweise (in Iterationen) in den Speicherplätzen der Objektdaten berechnet werden können ("in-place"-Verarbeitung). Näheres über die Faktorisierung und die Entwicklung von Signalflußgraphen findet sich in den Abschnitten 3.3 und 3.4.

Signaltransformationen dienen der Verarbeitung digitalisierter Signale. Ein wesentliches Problem hierbei ist die zur Transformation benötigte Rechenzeit. Wenn die Berechnung auf einem Einprozessorrechner abläuft, ist neben der Zykluszeit vor allem die Anzahl und die Art der erforderlichen Rechenoperationen sowie die Anzahl der benötigten Speicherzugriffe maßgebend. Falls die Transformation von einer Spezial-Hardware durchgeführt wird, ist zudem noch die Parallelisierbarkeit des Algorithmus' von entscheidender Bedeutung, da dieser dann u.U. in eine Vielzahl von parallel ablaufenden Operationen aufgeteilt werden kann.

Da die Anzahl der Objektdaten (Stützstellen) recht groß sein kann, wie z.B. in der digitalen Bildverarbeitung, interessiert hauptsächlich, wie sich der Rechenaufwand in Abhängigkeit von der Anzahl N der Datenelemente verhält. Es ist deshalb üblich, die Ordnung der Komplexität von Algorithmen in Form eines Wachstumsgesetzes abhängig von N anzugeben: $K=O(f(N))$. Hierbei kommt es nur auf die (Größen-) Ordnung an, weswegen man von der Art und der exakten Anzahl der Operationen absieht und sich ausschließlich dafür interessiert, nach welchem Gesetz der Rechenaufwand mit zunehmender Anzahl N der Datenelemente wächst. Für den allgemeinen Fall ist es sinnvoll und üblich, eine Multiplikation plus eine Addition als eine Operation zu definieren.

Für diskrete Transformationen nach den Summenformen (1.2) oder den Matrixformen (1.3) ist unmittelbar einleuchtend, daß für die Komplexität $K=O(N^2)$ gilt. In den Abschnitten 3.3 und 3.4 wird gezeigt, auf welche Weise unter bestimmten Umständen die Komplexität auf $O(N \cdot \log_p N)$ reduziert werden kann. Bei 512 Datenwerten (Anzahl der Bildpunkte pro Zeile eines normal aufgelösten Bildes) benötigt man dann nur 4608 Operationen anstelle von 262144, wenn der Radix p wie meist üblich als 2 gewählt wird.

1.4 Organisation des Buches

Nach dieser kompakten Einführung, in der Motivation und Anwendungen sowie die wichtigsten Begriffe dargelegt wurden, folgt zunächst ein Überblick über die in diesem Buch verwendeten Orthogonalsysteme. Dabei beginnen wir mit der Fourier-Transformation (FT), die ja eigentlich eine unitäre Transformation ist, d.h. einen komplexen Kern hat und auch komplexwertige Objektdaten zu verarbeiten gestattet. Allerdings sind die meisten Anwendungen der FT so gelagert, daß die Daten des Objektbereichs reell sind. In diesem Fall verhält sich die FT wie eine Orthogonaltransformation mit dem Unterschied, daß die Transformationskoeffizienten komplex sind. Über die Herleitung, die Eigenschaften und die Anwendungen der diskreten FT existiert eine umfangreiche Sammlung von Literatur. Wir beschränken uns deshalb bei der Darstellung der FT auf einige wichtige Definitionen und Theoreme.

Ebenfalls im 2. Kapitel werden dann weitere Transformationen mit trigonometrischen aber reellwertigen Basisfunktionen vorgestellt, nämlich die oft als reelle Fourier-Transformation bezeichnete Hartley-Transformation sowie die in der Bildverarbeitung häufig benutzten Cosinus- und Sinus-Transformationen. Neben diesen "trigonometrischen" Transformationen spielen in der Signalverarbeitung eine Reihe von Transformationen mit nichtsinusförmigen Basisfunktionen eine Rolle. Hierzu zählen die Systeme der mäanderförmigen Walsh-Funktionen sowie die Haar- und die Slant-Transformation. Da diese Transformationen weitaus weniger bekannt und beschrieben sind als die Fourier-Transformation wurde ihnen mehr Raum gewidmet.

Bevor wir uns mit der Algorithmierung eindimensionaler (Kapitel 4) und zweidimensionaler Transformationen (Kapitel 6) befassen, werden im Kapitel 3

allgemeine Gesichtpunkte für die Entwicklung der Algorithmen behandelt. Hierzu zählen die Berechnungsstrategien, die Faktorisierung der Transformationsmatrizen und die Anwendung der Restklassenarithmetik in den Algorithmen. Weiterhin wird die Darstellung der Algorithmen durch Signalflußgraphen vorgestellt. In engem Zusammenhang hiermit stehen Fragen der Reihenfolge von Eingangs- und Ausgangsdaten und der Reihenfolge der Iterationsschritte des Algorithmus'. Während bei der Vorstellung der Transformationen im Kapitel 2 die Fourier-Transformation ihrer Wichtigkeit und ihrer allgemeinen Bekanntheit wegen an erster Stelle stand, wird für die Kapitel 4 und 6 eine andere Reihenfolge gewählt. In diesen Kapiteln stellen wir jeweils die Algorithmen für die ein- bzw. zweidimensionale WHT zuerst vor. Für dieses Vorgehen gibt es zwei Gründe: Einerseits sind die WHT-Algorithmen wegen des Fehlens der Drehfaktoren einfacher als die der Fourier-Transformation. Zum anderen gibt es Algorithmen für die Fourier-Transformation, die über die Berechnung der WHT führen.

Bevor im Kapitel 6 Algorithmen zweidimensionaler Transformationen dargestellt werden, diskutieren wir verschiedene Punkte, die für den Entwurf zweidimensionaler Algorithmen von allgemeiner Bedeutung sind (Kapitel 5). Hierzu gehören Fragen der Separierbarkeit, der Rückführung auf eine eindimensionale Transformation sowie die Berechnung der Koeffizienten aus solchen niedrigerer Ordnung oder aus denen einer anderen (einfacheren) Transformation.

Hinweise auf die Implementierung der Algorithmen in Software und/oder Hardware werden schließlich im Kapitel 7 gegeben.

Zu jedem Kapitel werden Quellen und Übersichtsartikel sowie einschlägige Bücher genannt. Zusätzlich wird an dieser Stelle noch auf eine kommentierte Sammlung von bedeutenden Zeitschriftenartikeln (benchmark papers) hingewiesen [1.8].

2 Die wichtigsten diskreten Orthogonaltransformationen

In diesem Kapitel definieren wir einige für die Signalverarbeitung wichtige diskrete Transformationen. Die Darstellung ist absichtlich knapp gehalten, denn nicht die diskreten Transformationen an sich, sondern ihre effiziente Berechnung sind der wesentliche Gegenstand dieses Buches.

Zunächst werden Orthogonaltransformationen mit trigonometrischen Basisfunktionen (Fourier-, Hartley-, Cosinus- und Sinus-Transformation) vorgestellt. Wegen des geringeren Rechenaufwandes erfreuen sich Orthogonaltransformationen mit mäanderförmigen Basisfunktionen einer gewissen Beliebtheit in der Signalverarbeitung. Von ihnen werden die Walsh-Transformation (in Walsh-, Paley- und Hadamard-Ordnung), die Haar- und die Slant-Transformation definiert.

2.1 Fourier-Transformation

Die diskrete Fourier-Transformation (DFT) wird überlicherweise nicht durch Diskretisierung der Fourier-Transformation kontinuierlicher Signale erzeugt, vielmehr wird i.allg. von einer selbständigen Definition der Fourier-Transformation diskreter Signale endlicher Dauer ausgegangen. Man definiert deshalb die diskrete Fourier-Transformierte des Datensatzes $x(n)$ als $X_f(k)$ als

$$X_f(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{kn}, \qquad k = 0,1,\ldots,N-1. \tag{2.1}$$

Dabei ist $W_N = \exp(-2j\pi/N)$ die Basisfunktion der DFT und $j = \sqrt{-1}$. Die Exponentialfunktionen W_N^{kn} in (2.1) werden oft als Drehfaktoren bezeichnet. Sie sind orthogonal, denn es gilt

$$\sum_{m=0}^{N-1} W_N^{km}\, W_N^{nm} = \begin{cases} N & \text{wenn } k-n=0 \text{ oder } k-n=N, \\ 0 & \text{sonst.} \end{cases}$$

Schreibt man die Exponenten $E = k \cdot n$ der Basisfunktion W_N in Form einer Matrix, so erhält man für ihre Gesamtheit die folgende Matrix:

$$[E] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \ldots & 0 \\ 0 & 1 & 2 & 3 & 4 \ldots & N-1 \\ 0 & 2 & 4 & 6 & 8 \ldots & 2(N-1) \\ . & . & . & . & . \ldots & . \\ . & . & . & . & . \ldots & . \\ . & . & . & . & . \ldots & . \\ 0 & N-1 & 2(N-1) & . & . \ldots & (N-1)(N-1) \end{bmatrix}.$$

Die diskrete Fourier-Transformation ordnet einem Satz von N Daten x(n) einen gleichgroßen Satz von Fourier-Koeffizienten $X_f(k)$ zu. Diese Größen sind eineindeutig zugeordnet, d.h. man kann aus den Größen x(n) die diskreten Fourier-Koeffizienten $X_f(k)$ bestimmen und umgekehrt aus den diskreten Fourier-Koeffizienten die diskreten Funktionswerte berechnen. Die Berechnung der Funktionswerte x(n) aus den diskreten Fourier-Koeffizienten $X_f(k)$ heißt diskrete Fourier-Umkehrtransformation oder inverse Fourier-Transformation

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X_f(k)\, W_N^{-kn}, \qquad n = 0,1,\ldots,N-1.$$

Damit lautet das diskrete Fourier-Transformationspaar:

$$X_f(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{kn} \quad \longleftrightarrow \quad x(n) = N^{-1} \sum_{k=0}^{N-1} X_f(k)\, W_N^{-kn}. \tag{2.2}$$

In Matrixschreibweise haben wir folgende Darstellung:

$$\underline{X}_f = [W_N^E]\, \underline{x} \quad \longleftrightarrow \quad \underline{x} = N^{-1} [W_N^E]^{-1} \underline{X}_f$$

$$\text{mit } [W_N^E] = \begin{bmatrix} W_N^0 & W_N^0 & W_N^0 & \ldots & W_N^0 \\ W_N^0 & W_N^1 & W_N^2 & \ldots & W_N^{N-1} \\ . & . & . & \ldots & . \\ . & . & . & \ldots & . \\ W_N^0 & W_N^{N-1} & W_N^{2(N-1)} & \ldots & W_N^{(N-1)(N-1)} \end{bmatrix}.$$

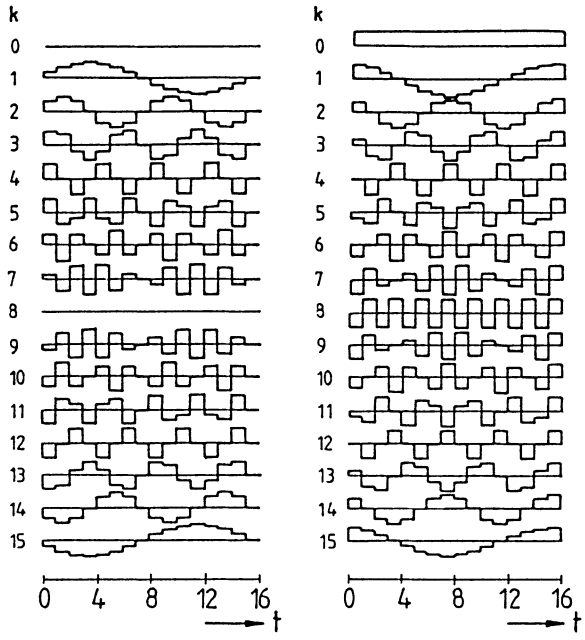


Bild 2.1. Quantisierte Basisfunktionen der Fourier-Transformation für N=8, nach [2.17]

Bild 2.1 stellt die ersten 16 Basisfunktionen der Fourier-Transformation in quantisierter Form dar.

Die Fourier-Transformation hat folgende wichtige Eigenschaften [2.1], die hier ohne Beweise zitiert werden, wobei $X_f(k)$ und $x(n)$ ein diskretes Fourier-Transformationspaar bilden.

(1) Linearität:
Die DFT ist eine lineare Transformation, d.h. wenn

$$x(n) = a \cdot x_1(n) + b \cdot x_2(n),$$

dann gilt

$$X_f(k) = a \cdot X_{f1}(k) + b \cdot X_{f2}(k)$$

mit $x_i(n) \longleftrightarrow X_{fi}(k)$ und $x(n) \longleftrightarrow X_f(k)$, wobei a und b reelle Zahlen sind und $i \in \{1,2\}$.

(2) Symmetrie:
Es gilt folgende Eigenschaft:

$$X_f(k) \longleftrightarrow x(-k).$$

(3) Verschiebung im Objektbereich:
Verschiebt man x(n) um die ganze Zahl m, so gilt

$$x(n-m) \longleftrightarrow X_f(k)W_N^{km}$$

(4) Verschiebung im Transformationsbereich:
Verschiebt man $X_f(k)$ um die ganze Zahl m, so entspricht das einer Multiplikation ihrer inversen diskreten Fourier-Transformierten mit dem Faktor W_N^{-kn}:

$$x(n)W_N^{-mn} \longleftrightarrow X_f(k-m).$$

(5) Faltungstheorem für den Frequenzbereich:
Wenn die Fourier-Transformationspaare

$$x_1(n) \longleftrightarrow X_{f1}(k) \quad \text{und} \quad x_2(n) \longleftrightarrow X_{f2}(k)$$

existieren, ist ihre Faltung gegeben durch

$$X_{f1}(k)\cdot X_{f2}(k) = \sum_{n=0}^{N-1} [x_1(n)\cdot x_2(k-n)], \qquad n = 0,1,\ldots,N-1.$$

(6) Diskrete Korrelation:
Die diskrete Korrelation wird durch das folgende Transformationspaar beschrieben:

$$\sum_{k=0}^{N-1} [x_1(n)\cdot x_2(k+n)] \longleftrightarrow X_{f1}^*(k)\cdot X_{f2}(k).$$

(7) Alternative Inversionsbeziehung:
Die inverse diskrete Fourier-Transformation läßt sich auch in folgender Form schreiben:

$$x(n) = N^{-1} \left[\sum_{k=0}^{N-1} X_f^*(k) \cdot W_N^{kn} \right]^* .$$

Diese Beziehung erlaubt es, sowohl für die (Vorwärts-) Transformation als auch für die inverse Transformation das gleiche Rechnerprogramm zu verwenden.

Die Fourier-Transformation ist eine unitäre Transformation: die Elemente x(n) können komplex sein. Für viele Anwendungen in der Signalverarbeitung sind die Daten des Objektbereichs jedoch reell, d.h. die Imaginärteile der x(n) sind identisch null. Deshalb bleiben Teile der vom Algorithmus vorgegebenen Operationen unbenutzt. Man beseitigt diese Redundanz entweder dadurch, daß man die x(n) für gerade und ungerade Werte von n separiert und als Real- und Imaginärteile einer FFT mit N/2 Daten auffaßt, oder daß zwei reelle Funktionen der Länge N gleichzeitig (als Real- und Imaginärteil) verarbeitet werden. Näheres hierzu findet man in den Abschnitten 4.2.7 und 6.2.6. Die Fourier-Transformation ist auf einem Ring von Einheitswurzeln definiert. Für reelle Daten muß man deshalb im Körper der komplexen Zahlen operieren. Man kann jedoch auch in endlichen Körpern und in Ringen von Polynomen arbeiten. Für spezielle Anwendungen kann man dadurch Vereinfachungen in der Berechnung erzielen. Eine für die diskrete Faltung geeignete Transformation erhält man z.B. wenn man die DFT in Ringen von Polynomen definiert [2.2], [2.3]. Bei dieser Polynom-Transformation ersetzt man den komplexen Exponentialausdruck durch eine Potenz der Variablen z und führt alle Operationen modulo eines Polynoms P(z) aus (vgl. Abschnitt 3.2):

$$X_f(k,z) = \sum_{n=0}^{p-1} x(n,z)\ z^{nk} \bmod P(z)$$

Dabei ist p eine Primzahl. Die inverse Transformation ist entsprechend als

$$x(n,z) = p^{-1} \sum_{k=0}^{p-1} X_f(k,z)\ z^{-nk} \bmod P(z).$$

definiert. Der Vorteil der Polynom-Transformation liegt in der geringeren Anzahl von Multiplikationen verglichen mit den in Kapitel 4 beschriebenen Algorithmen. Als Nachteil ist der höhere programmtechnische Aufwand zu nennen.

Ein andere Modifikation der DFT ist die zahlentheoretische Transformation (NTT) [2.2], [2.4]. Hierbei sind die Objektdaten Elemente des endlichen Restklassenrings der ganzen Zahlen modulo m (vgl. Abschnitt 3.2). Das Transformationspaar ist definiert als

$$X_f(k) = \sum_{n=0}^{N-1} (x(n)\cdot a^{nk}) \bmod m \quad \longleftrightarrow \quad x(n) = N^{-1} \sum_{k=0}^{N-1} (X_f(k)\cdot a^{-nk}) \bmod m$$

mit $n, k = 0,1,\ldots,N-1$ und $a \in Z$.

Dabei sind die Beziehungen $a^N \equiv 1 \bmod m$ und $GGT(a^n-1,m) = 1$ zu erfüllen. Hierdurch ist eine Einschränkung in der Wahl der Parameter a, N und m gegeben. Die NTT erlaubt nicht nur eine rein reelle Arithmetik, sondern es können bei geeigneter Parameterwahl Multiplikationen durch Verschiebeoperationen ersetzt werden. Als nachteilig erweist sich allerdings des Überlaufproblem, das sich aus dem begrenzten Zahlenbereich ergibt.

Da die Polynom-Transformation und die NTT bisher in der Praxis wenig Verwendung gefunden haben, gehen wir nicht weiter auf Algorithmen zu ihrer Durchführung ein.

2.2 Hartley-Transformation

Im Jahre 1942 veröffentlichte Hartley eine Integraltransformation ähnlich der Fourier-Transformation, jedoch mit reellem trigonometrischen Kern. Die diskrete Hartley-Transformation wurde von Bracewell [2.5] angegeben. Sie bietet eine weitere Möglichkeit, die Koeffizienten der DFT einer reellen Datenfolge effizient zu berechnen.

Für den diskreten Datensatz x(n) ist die diskrete Hartley-Transformation definiert als

$$X_{HY}(k) = \sum_{n=0}^{N-1} x(n)\,\mathrm{cas}(2\pi kn/N), \qquad k = 0,1,\ldots,N-1, \qquad (2.3)$$

wobei $\mathrm{cas}(\alpha) = \cos(\alpha) + \sin(\alpha)$. Die inverse Transformation ist bis auf den Faktor N^{-1} identisch mit (2.3):

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X_{HY}(k)\ \mathrm{cas}(2\pi kn/N) \qquad n = 0,1,\ldots,N-1.$$

Die Symmetrie des Transformationspaars ist eine wertvolle Eigenschaft der diskreten Hartley-Transformation (DHYT).

Man beachte, daß sich die DHYT von der Vorwärtstransformation der diskreten Fourier-Transformation lediglich durch Abwesenheit des Multiplikators "-j" des Sinusanteils unterscheidet. Dies bedeutet, daß die DHYT äquivalent der diskreten Fourier-Transformation einer reellwertigen Sequenz ist, wenn man den Imaginärteil der DFT von ihrem Realteil subtrahiert:

$$X_{HY}(k) = Re\{DFT[x(n)]\} - Im\{DFT[x(n)]\}.$$

Weil der Realteil der diskreten Fourier-Transformierten einer reellwertigen Sequenz gerade und der imaginäre Teil ungerade ist, kann die diskrete Fourier-Transformierte einfach aus der diskreten Hartley-Transformierten berechnet werden:

$$Re\{DFT[x(n)]\} = (1/2)\ \{\ DHYT[x(N-n)] + DHYT[x(n)]\ \},$$

$$Im\{DFT[x(n)]\} = (1/2)\ \{\ DHYT[x(N-n)] - DHYT[x(n)]\ \}.$$

Die Eigenschaften der diskreten Hartley-Transformation lassen sich unter Verwendung der DFT-Eigenschaften und dieser Beziehung einfach beweisen. Viele Eigenschaften der DHYT sind identisch den entsprechende Theoremen für die DFT [2.5].

Eine der wichtigsten Eigenschaften ist das Verschiebungstheorem:

$$x(n+m) \longleftrightarrow X_{HY}(k)\cos(2\pi mk/N) - X_{HY}(N-k)\sin(2\pi mk/N).$$

Die diskrete Hartley-Transformation besitzt außerdem die folgende Faltungs-Eigenschaft:

$$x_1(n)\cdot x_2(n)$$

$$\updownarrow$$

$$(1/2)[X_{HY1}(k)X_{HY2}(k)+X_{HY1}(k)X_{HY2}(N-k)+X_{HY1}(N-k)X_{HY2}(k)-X_{HY1}(N-k)X_{HY2}(N-k)].$$

Die o.g. Eigenschaften erlauben die Entwicklung eines schnellen Hartley-Transformationsalgorithmus'. Verschiedene Algorithmen zur Berechnung der

Hartley-Transformation und die entsprechenden Signalflußgraphen sind Gegenstand des Abschnitts 4.3.

2.3 Cosinus- und Sinus-Transformation

Die diskrete Cosinus-Transformation (DCT) eines reellen Datensatzes x(n) und ihre Inverse (IDCT) sind wie folgt definiert [2.6]:

$$X_c(k) = 2c(k) \sum_{n=0}^{N-1} x(n)\cos((2n+1)k\pi/(2N)), \qquad k = 0,1,\ldots,N-1,$$

$$\updownarrow$$

$$x(n) = N^{-1} \sum_{k=0}^{N-1} c(k)\, X_c(k)\cos((2n+1)k\pi/(2N)), \qquad n = 0,1,\ldots,N-1.$$

Dabei ist $c(k) = \begin{cases} 1/\sqrt{2} & \text{für } k=0 \\ 1 & \text{für } k = 1,2,\ldots,N-1. \end{cases}$

Die Menge der Basisfunktionen ist eine Teilmenge der diskreten Chebyshev-Polynome [2.7].

Die Cosinus-Transformation ist in der Signalverarbeitung deswegen interessant, weil sie bezüglich der Dekorrelation ihrer Koeffizienten besonders günstige Eigenschaften hat, sofern sich das Signal als Markov-Prozess modellieren läßt. Außerdem besitzt sie die Eigenschaft höchster Energiekompaktheit mehr als irgendeine andere Transformation mit einem schnellen Berechnungsalgorithmus. Bezüglich der Konzentration der Energie in wenigen Koeffizienten wird sie nur von der Karhunen-Loève-Transformation übertroffen, für die jedoch kein schneller Algorithmus existiert. Außerdem kann man für die Cosinus-Transformation eine Beziehung zwischen zyklischer Faltung und Multiplikation im Transformationsbereich angeben [2.8]. Die DCT hat zahlreiche Anwendungen bei der Verarbeitung von Sprachsignalen und in der Bildverarbeitung gefunden. Ihre ersten 16 Basisfunktionen sind im Bild 2.2 dargestellt.

Die diskrete Sinus-Transformation (DST) wurde von Jain [2.9], [2.18] zuerst angegeben und anschießend von Kekre und Solanki [2.10] wie folgt definiert:

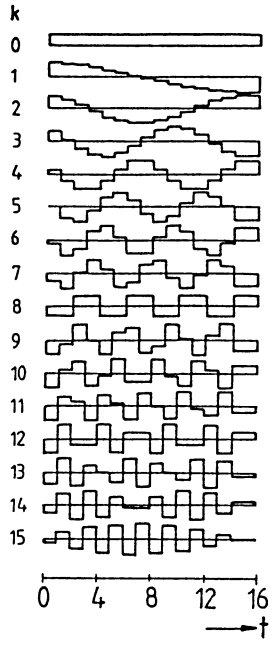


Bild 2.2. Quantisierte Basisfunktionen der Cosinus-Transformation für N=8, nach [2.17]

Für eine Datensequenz $\{x(n)\} = \{x(1),x(2),\ldots,x(N)\}$ ist

$$X_s(k) = s(k) \sum_{n=1}^{N} x(n)\sin(\pi nk/(N+1)), \qquad k = 1,2,\ldots,N,$$

wobei

$$s(k) = \begin{cases} \sqrt{2/(N+1)} & \text{für } k = 1,2,\ldots,N \\ 0 & \text{für } k=0 \text{ oder } k \geq (N+1). \end{cases}$$

Diese Definition wird in diesem Buch als DST(I) bezeichnet. Die Basismenge der DST(I) lautet:

$$\Psi_{nk} = \sqrt{2/(N+1)}\ \sin(\pi nk/(N+1)) = s(k)\ \sin(\pi nk/(N+1)) \qquad n, k = 1,2,\ldots,N.$$

Andererseits definierten Kekre und Solanki die DST als:

$$\Psi_{nk} = \sqrt{2/N}\ \sin(\pi(n+1)(2k+1)/(2N)), \qquad n, k = 0,1,\ldots,N-1.$$

Diese Definition bezeichnen wir als DST(II).

Die Rücktransformation der DST(I) bzw. DST(II) gestaltet sich wie folgt:

$$x(n) = \sum_{k=1}^{N} X_s(k) \cdot \Psi_{nk}.$$

Die DCT und die diskrete Sinus-Transformation (DST) stehen in enger Beziehung zur diskreten Fourier-Transformation. Der Zusammenhang zwischen der Berechnung der DFT, der DCT und der DST wird in den Abschnitten 4.4.1 bis 4.4.3 und in 4.5.2 diskutiert.

2.4 Walsh-Hadamard-Transformation

Die diskrete Walsh- oder Walsh-Hadamard-Transformation (WHT) wird häufig zur Lösung von Problemen der digitalen Signalverarbeitung herangezogen. Die Vorteile der WHT liegen hauptsächlich in den einfachen hardware- und softwaremäßigen Implementierungsmöglichkeiten. Da ihre Basisfunktionen nur die Werte +1 und −1 annehmen, entfallen die kosten- und zeitintensiven Multiplikationen. Das ist vom Standpunkt des Rechaufwandes ein Vorteil, jedoch gibt es auch Anwendungsfälle, für die rechteckförmige Funktionen mit mehr als zwei Funktionswerten zweckmäßig sind. Deshalb sind verschiedene Ansätze gemacht worden, die Walsh-Funktionen derart zu verallgemeinern, daß orthogonale und vollständige Funktionensysteme mit 3 oder mehr Funktionswerten entstehen.

Ein solches System ist das der Chrestenson-Funktionen [2.11], das komplexwertige Funktionen enthält. Diese sind die Basisfunktionen der unitären Chrestenson-Transformation. Wegen der Tatsache, daß die Chrestenson-Funktionen komplex sind und sich deshalb für manche Anwendungen als unzweckmäßig erweisen, wurden verschiedene reelle Varianten entwickelt [2.12], [2.13], [2.14]. Im Rahmen dieses Buches kann jedoch auf die verallgemeinerten Walsh-Funktionen nicht eingegangen werden. Es sei deshalb auf die o.g. Literaturstellen verwiesen.

Es gibt vier Versionen der Walsh-Hadamard-Transformation (WHT), die sich durch die Art der Anordnung der Basisfunktionen unterscheiden: Die nach Walsh (oder nach steigenden Sequenzen) geordnete Transformation $(WHT)_W$, die nach Hadamard (oder natürlich) geordnete Transformation $(WHT)_H$, die nach Pa-

ley (oder dyadisch) geordnete Transformation $(WHT)_p$, und die nach geraden und ungeraden Funktionen geordnete Transformation $(WHT)_{cs}$.

Die Basisfunktionen für die WHT sind die Walsh-Funktionen. Definition und Eigenschaften der Walsh-Funktionen werden wie folgt angegeben [1.1], [1.6], [2.15], [2.16]:

Die periodischen, *kontinuierlich* definierten Walsh-Funktionen $wal_w(k,t)$ mit der Periode N (k = 0,1,...) seien im fundamentalen Intervall [0,N] durch die Rekursionsgleichungen

$$wal_w(0,t) = 1, \qquad t \in [0,N]$$

$$wal_w(2k-1,t) = \begin{cases} wal_w(k-1,2t) & t \in [0,N/2] \\ (-1)^k wal_w(k-1,2t-N) & t \in [N/2,N] \end{cases}$$

und

$$wal_w(2k,t) = \begin{cases} wal_w(k,2t) & t \in [0,N/2] \\ (-1)^k wal_w(k,2t-N) & t \in [N/2,N] \end{cases}$$

gegeben.

Analog zur DFT wird die *diskrete* Walsh-Transformation als Transformation einer Folge von Abtastwerten eines Signals mit einer Basis aus Abtastwerten von Walsh-Funktionen definiert:

$$X_w(k) = \sum_{n=0}^{N-1} x(n) \cdot Wal_w(k,n), \qquad k = 0,1,2,\ldots,N-1.$$

Die inverse Beziehung lautet:

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X_w(k) \cdot Wal_w(k,n), \qquad n = 0,1,2,\ldots,N-1.$$

a) Nach Walsh oder nach Sequenzen geordnete Walsh-Funktionen: $Wal_w(k,n)$

Die Anzahl N der Abtastwerte sei gleich 2^r und r = 1,2,3,... . Dann kann man die Walsh-Funktionen in der Form

$$Wal_w(k,n) = (-1)^{\sum_{j=0}^{r-1} (U_j + U_{j+1}) V_{r-1-j}} \qquad \text{für } k,\ n = 0,1,\ldots,N-1 \qquad (2.4)$$

definieren, wobei für die nichtnegativen ganzen Zahlen k und n folgende eindeutige Dualzahldarstellungen gelten:

$$k = \sum_{m=0}^{r-1} U_m 2^m, \qquad \text{d.h. } (k)_{dez} = (U_{r-1}\ U_{r-2} \ldots U_1\ U_0)_{bin},$$

$$n = \sum_{m=0}^{r-1} V_m 2^m, \qquad \text{d.h. } (n)_{dez} = (V_{r-1}\ V_{r-2} \ldots V_1\ V_0)_{bin}.$$

$$U_m, V_m \in \{0,1\} \qquad (2.5)$$

Die durch (2.4) definierte Ordnung heißt natürliche Ordnung oder Walsh-Ordnung.

Die Walsh-Funktionen besitzen die folgende Symmetrieeigenschaft:

$$Wal_w(k,n) = Wal_w(n,k), \qquad \text{für } n,\ k = 0,1,\ldots,N-1.$$

Das Produkt zweier Walsh-Funktionen ergibt wieder eine Walsh-Funktion, insbesondere gilt

$$Wal_w(k,n) \cdot Wal_w(m,n) = Wal_w(k \oplus m, n)$$

für $k,\ m = 0,1,2\ldots$ und $n = 0,1,\ldots,N-1$.

Hierbei symbolisiert das Zeichen "$\oplus$" die Modulo-2-Addition (Dualzahladdition ohne Übertrag). Sei

$$k = \sum_{c \geq 0} U_c 2^c, \qquad m = \sum_{c \geq 0} Q_c 2^c \qquad \text{und} \qquad U_c, Q_c \in \{0,1\},$$

so ist $k \oplus m = \sum_{c \geq 0} \delta_c 2^c$

mit $\delta_c = \begin{cases} 1 & \text{falls } U_c + Q_c = 1 \\ 0 & \text{sonst.} \end{cases}$

Darüber hinaus erfüllen die Walsh-Funktionen die diskrete Orthogonalitätsbeziehung:

$$N^{-1} \sum_{n=0}^{N-1} Wal_w(k,n)Wal_w(m,n) = \delta_{km}, \qquad k, m = 0,1,2,\ldots,N-1,$$

$$\delta_{km} = \begin{cases} 1 & \text{falls } k = m \\ 0 & \text{wenn } k \neq m. \end{cases}$$

Die Basisfunktionen der sequenzgeordneten WHT ergeben sich somit wie im Bild 2.3 dargestellt ist. Dabei ist θ ein normalisiertes Argument.

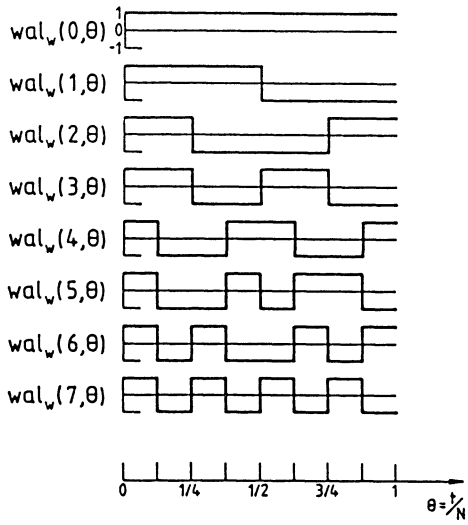


Bild 2.3. Nach Walsh geordnete kontinuierliche Walsh-Funktionen für N=8, nach [2.7]

b) Nach Paley oder dyadisch geordnete Walsh-Funktionen: $Wal_p(k,n)$

Walsh-Funktionen sind Elemente der dyadischen Gruppe und können durch den Gray-Code geordnet werden [1.1], [1.2]. Ordnet man die nach ansteigenden Sequenzen numerierten Walsh-Funktionen geeignet um, so kann man über einen Radix-2-Algorithmus für die Berechnung der $N=2^r$ diskreten Koeffizienten die Anzahl der notwendigen Additionen auf $r \cdot 2^r$ (statt $2^r(2^r-1)$) reduzieren (vgl. Abschnitt 3.4). Für die Umordnung benötigt man die Gray-Code-Konvertierung. Jeder nichtnegativen ganzen Zahl $k < 2^r$ mit der Dualzahldarstellung aus (2.5) ist die nichtnegative ganze Zahl $G(k) < 2^r$ mit der Dualdarstellung

$$G(k) = \sum_{v=0}^{r-1} (U_v \oplus U_{v+1}) 2^v, \qquad U_r = 0 \tag{2.6}$$

umkehrbar eindeutig zugeordnet. Die Umkehrung $G^{-1}(k)$ erfolgt gemäß (2.5) mit

$$U_v = \begin{cases} 1 & \text{für } U_{v+1} + U_{v+2} + \ldots + U_r \text{ ungerade} \\ 0 & \text{sonst.} \end{cases}$$

Die nach Paley (oder dyadisch) geordneten Walsh-Funktionen $wal_p(k,t)$ sind gegeben durch die Beziehung $wal_p(G(k),t) = wal_w(k,t)$ mit $k = 0,1,2\ldots$.

Für nichtnegative Zahlen $k,n < N=2^r$ mit den Dualzahldarstellungen aus (2.4) sind dann wegen (2.5) und (2.6) die Werte der nach Paley geordneten Walsh-Funktionen in den Punkten n gegeben durch

$$Wal_p(k,n) = (-1)^{\sum_{j=0}^{r-1} U_j V_{r-1-j}} . \tag{2.7}$$

Diese Funktionen erfüllen ebenfalls die Symmetriebedingung

$$Wal_p(k,n) = Wal_p(n,k).$$

Die nach Paley geordneten Walsh-Funktionen sind für N=8 im Bild 2.4 dargestellt.

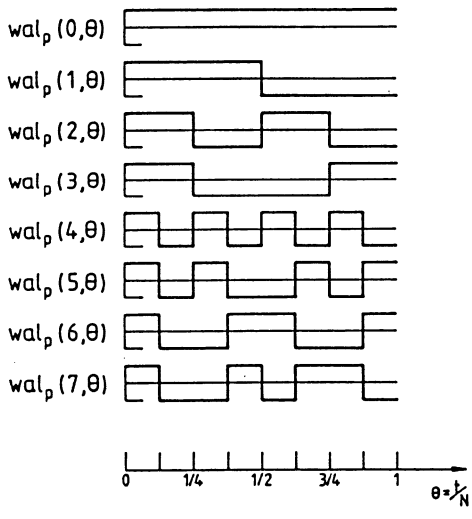


Bild 2.4. Nach Paley geordnete kontinuierliche Walsh-Funktionen für N=8, nach [2.7]

Die Beziehung zwischen $wal_p(k,t)$ und $wal_w(k,t)$ kann wie folgt beschrieben werden [2.7]:

$$wal_p(k,t) = wal_w(G^{-1}(k),t). \tag{2.8}$$

c) Nach Hadamard oder natürlich geordnete Walsh-Funktionen : $Wal_H(k,n)$

Eine weitere Vereinfachung bei der Berechnung der Koeffizienten erreicht man, wenn man die nach Paley geordneten Walsh-Funktionen mit Hilfe der Bitumkehroperation R(k) weiter umordnet. Diese ordnet jeder nichtnegativen Zahl $k < 2^r$ mit der Dualzahldarstellung

$$k = \sum_{j=0}^{r-1} U_j 2^j, \qquad U_j \in \{0,1\}$$

die nichtnegative ganze Zahl $R(k) < 2^r$ mit der Dualzahldarstellung

$$R(k) = \sum_{j=0}^{r-1} U_{r-1-j} 2^j$$

zu. Es ist offenkundig, daß R(R(k)) = k gilt.

Zur Umordnung eines Vektors $\underline{x}(n)$ stellt man also den Index n als Binärzahl dar und kehrt dann die Reihenfolge der Bits um. Bild 2.5 zeigt ein Beispiel

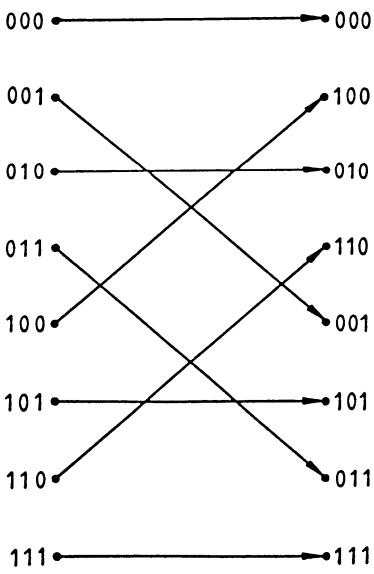


Bild 2.5. Beispiel zur Bitumkehroperation für N=8

für die Bitumkehroperation. Die Koeffizienten X(k) und X(i) werden miteinander vertauscht, wenn sich die ganze Zahl i durch die Bitumkehroperation aus der ganzen Zahl k ergibt. Diese Art der Bitumkehrsortierung wird in diesem Buch auch als BRO = BEO(1) bezeichnet (vgl. Abschnitt 3.4).

Die auf diese Weise geordneten Walsh-Funktionen $wal_h(k,t)$ heißen nach Hadamard (oder natürlich) geordnet. Sie ergeben sich aus den nach Paley geordneten Walsh-Funktionen gemäß $wal_h(k,t) = wal_p(R(k),t)$. Zu den nach ansteigenden Sequenzen geordneten Walsh-Funktionen stehen sie über

$$wal_h(R(G(k)),t) = wal_w(k,t)$$

in Beziehung. Umgekehrt gilt

$$wal_h(k,t) = wal_w(G^{-1}(R(k)),t). \tag{2.9}$$

Die nach Hadamard geordneten Walsh-Funktionen sind im Bild 2.6 dargestellt. Für nichtnegative ganze Zahlen k, $n < N = 2^r$ mit der Dualzahldarstellung (2.5) sind dann die Werte der nach Hadamard geordneten Walsh-Funktionen in den Punkten n gegeben durch

$$Wal_H(k,n) = (-1)^{\sum_{j=0}^{r-1} U_j V_j}. \tag{2.10}$$

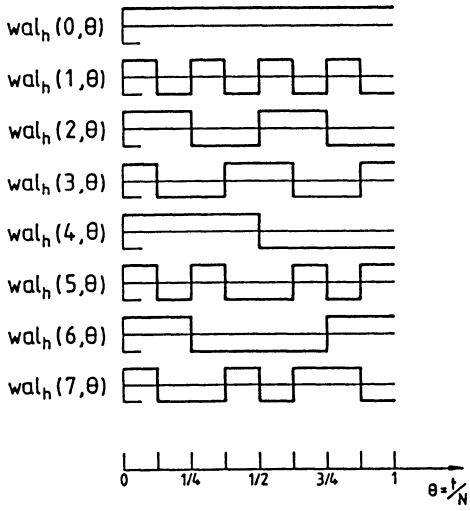


Bild 2.6. Nach Hadamard geordnete kontinuierliche Walsh-Funktionen für N=8, nach [2.7]

Diese Funktionen haben deshalb ebenfalls die Symmetrieeigenschaft

$$Wal_H(k,n) = Wal_H(n,k).$$

d) Walsh-Funktionen in Cal/Sal-Ordnung: $Wal_{CS}(k,n)$

Sei $\{k\}$ eine Sequenz von $wal_w(k,t)$, dann wird $\{k\}$ folgendermaßen dargestellt:

$$\{k\} = \begin{cases} 0 & \text{für } k=0, \\ k/2 & \text{für } k \text{ gerade}, \\ (k+1)/2 & \text{für } k \text{ ungerade}. \end{cases}$$

Die Cal- und Sal-Funktionen beziehen sich auf $wal_w(k,t)$:

$$cal(\{k\},t) = wal_w(k,t) \qquad \text{für } k \text{ gerade}, \tag{2.11}$$

$$sal(\{k\},t) = wal_w(k,t) \qquad \text{für } k \text{ ungerade}. \tag{2.12}$$

Bild 2.7 stellt die nach dem Cal/Sal-System geordneten Walsh-Funktionen im Vergleich mit den trigonometrischen Funktionen dar.

Die folgende Zusammenstellung zeigt die Beziehungen der natürlich geordneten Walsh-Funktionen $wal_h(k,t)$ und der dyadisch geordneten Funktionen $wal_p(k,t)$ zu den nach ansteigenden Sequenzen geordneten Funktionen $wal_w(k,t)$ (vgl. (2.9) und (2.8)):

$$wal_h(k,t) = wal_w(G^{-1}(R(k)),t), \qquad wal_p(k,t) = wal_w(G^{-1}(k),t).$$

Tabelle 2.1 stellt die Beziehungen von k, R(k), $G^{-1}(R(k))$ und $G^{-1}(k)$ für r=3, d.h. N=8 dar.

Den oben definierten Basisfunktionen gemäß werden nun die verschiedenen Walsh-Hadamard-Transformationen in Matrixdarstellung überführt:

a) Nach Walsh (oder Sequenzen) geordnete $(WHT)_w$

Die $(WHT)_w$ einer Datensequenz $\{x(n)\}$ wird in Matrixdarstellung als

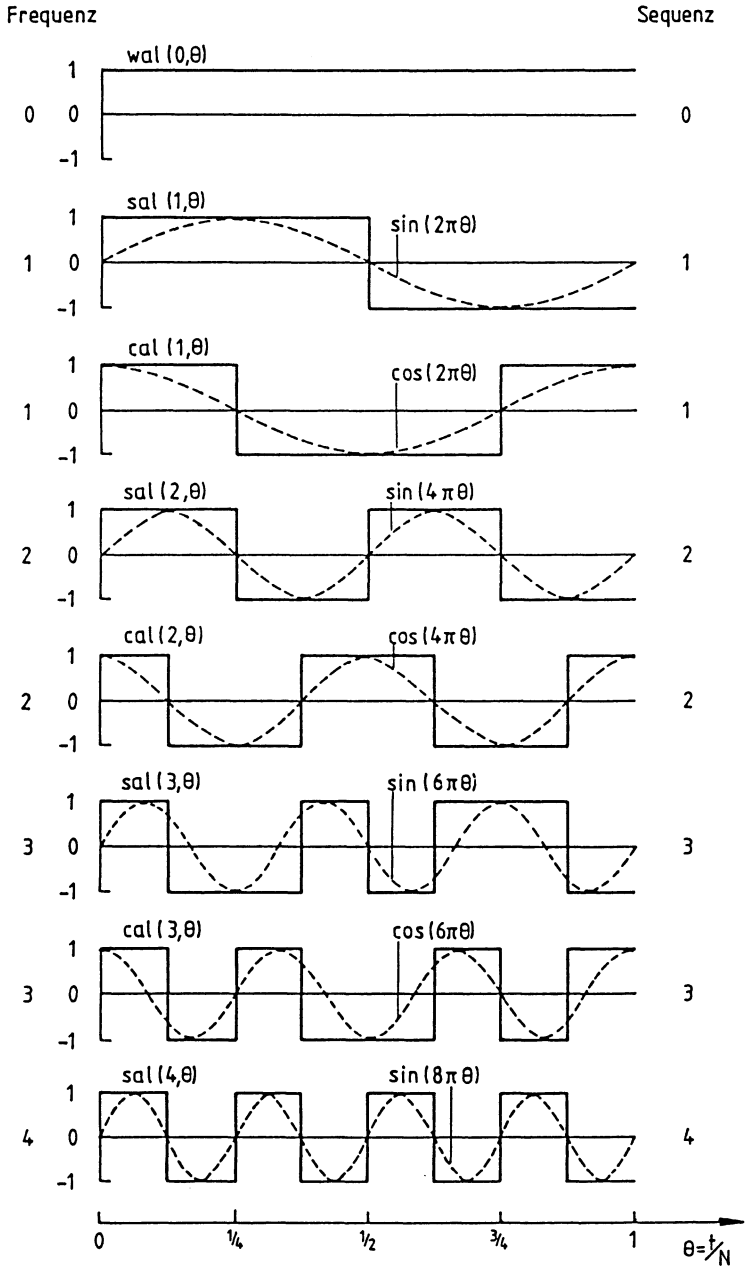


Bild 2.7. Walsh-Funktionen in Cal-Sal-Ordnung für N=8 im Vergleich mit trigonometrischen Funktionen, nach [2.7]

Tabelle 2.1. Beziehungen zwischen k, R(k), $G^{-1}(R(k))$ und $G^{-1}(k)$ für r=3

k		R(k)		$G^{-1}(R(k))$		$G^{-1}(k)$	
binär	dezimal	binär	dezimal	binär	dezimal	binär	dezimal
000	0	000	0	000	0	000	0
001	1	100	4	111	7	001	1
010	2	010	2	011	3	011	3
011	3	110	6	100	4	010	2
100	4	001	1	001	1	111	7
101	5	101	5	110	6	110	6
110	6	011	3	010	2	100	4
111	7	111	7	101	5	101	5

$$\underline{X}_W = [H_W(L)]\,\underline{x}, \qquad L = 1,2,\ldots,r \qquad \text{und} \qquad N = 2^r$$

definiert, wobei $[H_W(L)]$ eine nach Walsh geordnete Transformationsmatrix der Ordnung N×N ist. Die Elemente von $[H_W(L)]$ sind nach (2.4) und (2.5)

$$h_W(k,n) = (-1)^{\sum_{j=0}^{r-1} V_{r-1-j}(U_j+U_{j+1})} \qquad k,\ n = 0,1,\ldots,N-1.$$

Für N=8 ergibt sich z.B.:

$$[H_W(3)] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix} \quad \begin{matrix} \text{Sequenz} \\ 0 \\ 1 \\ 1 \\ 2 \\ 2 \\ 3 \\ 3 \\ 4 \end{matrix}$$

$[H_W(L)]$ ist orthogonal und symmetrisch. Deshalb ist die inverse Transformation $(IWHT)_W$ gegeben durch

$$\underline{x} = N^{-1}[H_w(L)]\ \underline{X}_w, \qquad L = 1,2,\ldots,r.$$

Insgesamt lautet das diskrete $(WHT)_w$-Paar in Summendarstellung:

$$X_w(k) = \sum_{n=0}^{N-1} x(n)h_w(k,n) \quad <\!\!\longrightarrow \quad x(n) = N^{-1}\sum_{k=0}^{N-1} X_w(k)h_w(k,n),$$

$$k = 0,1,\ldots,N-1, \qquad n = 0,1,\ldots,N-1.$$

b) Nach Paley (oder dyadisch) geordnete $(WHT)_p$

Ähnlich der $(WHT)_w$ wird die $(WHT)_p$ definiert als

$$\underline{X}_{wp} = [H_p(L)]\ \underline{x}, \qquad L = 1,2,\ldots,r.$$

Für die Elemente von $[H_p(L)]$ gilt nach (2.7)

$$h_p(k,n) = (-1)^{\sum_{j=0}^{r-1} U_{r-1-j}V_j}, \qquad k, n = 0,1,\ldots,N-1.$$

Beispiel mit N=8:

$$[H_p(3)] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ & & & & & & & \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{matrix} \text{Sequenz} \\ 0 \\ 1 \\ 2 \\ 1 \\ \\ 4 \\ 3 \\ 2 \\ 3 \end{matrix}.$$

Die inverse Transformation ist

$$\underline{x} = N^{-1}[H_p(L)]\ \underline{X}_{wp}, \qquad L = 1,2,\ldots,r.$$

Das diskrete $(WHT)_p$-Paar läßt sich in Summenform wie folgt schreiben:

$$X_{WP}(k) = \sum_{n=0}^{N-1} x(n)h_P(k,n) \quad <\!\!\longrightarrow\!\!> \quad x(n) = N^{-1} \sum_{k=0}^{N-1} X_{WP}(k)h_P(k,n),$$

$$k = 0,1,\ldots,N-1, \qquad n = 0,1,\ldots,N-1.$$

c) Nach Hadamard (oder natürlich) geordnete $(WHT)_H$

Die $(WHT)_H$ der Sequenz $\{x(n)\}$ ist als

$$\underline{X}_{WH} = [H_H(L)]\ \underline{x}, \qquad L = 1,2,\ldots,r.$$

definiert, wobei $\underline{X}_H$ die $(WHT)_H$-Koeffizientenmatrix ist. Die inverse Transformation stellt sich als

$$\underline{x} = N^{-1}\ [H_H(L)]\ \underline{X}_{WH}$$

dar. Die Elemente von $[H_H(L)]$ sind nach (2.10):

$$h_H(k,n) = (-1)^{\sum_{j=0}^{r-1} V_j U_j}, \qquad k, n = 0,1,\ldots,N-1.$$

Für N=8 gilt z.B.:

$$[H_H(3)] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \quad \begin{matrix} \text{Sequenz} \\ 0 \\ 4 \\ 2 \\ 2 \\ \\ 1 \\ 3 \\ 1 \\ 3 \end{matrix}$$

Es zeigt sich, daß $[H_H(L)]$ rekursiv definiert werden kann:

$$[H_H(L)] = \begin{bmatrix} [H_H(L-1)] & [H_H(L-1)] \\ [H_H(L-1)] & -[H_H(L-1)] \end{bmatrix}$$

$$L = 1,2,\ldots,r, \qquad r=\log_2 N \qquad \text{und} \qquad [H_H(0)]=1.$$

Für das $(WHT)_H$-Paar erhalten wir:

$$X_{WH}(k) = \sum_{n=0}^{N-1} x(n)h_H(k,n) \quad <\!\!\longrightarrow\!\!> \quad x(n) = N^{-1} \sum_{k=0}^{N-1} X_{WH}(k)h_H(k,n),$$

$$k = 0,1,\ldots,N-1, \qquad n = 0,1,\ldots,N-1.$$

d) Nach Cal- und Sal-Funktionen geordnete $(WHT)_{cs}$

Die ersten 16 Walsh-Funktionen in Cal/Sal-Ordnung sind im Bild 2.7 dargestellt. Ihr Zusammenhang mit $wal_w(k,t)$ ist bereits in (2.11) und in (2.12) angegeben. Man kann dies auch folgendermaßen formulieren:

$$wal_w(k,t) = \begin{cases} wal_{cs}((1/2)k,t) & \text{für } k = 0,2,4,\ldots,N-2, \\ wal_{cs}(N-(1/2)(k+1),t) & \text{für } k = 1,3,5,\ldots,N-1. \end{cases}$$

Diese diskrete Version führt zu den Walsh-Matrizen in Cal/Sal-Ordnung

$$[H_{cs}(1)] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

$$[H_{cs}(2)] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix} \begin{matrix} Wal_w(0,n) \\ Cal(1,n) \\ Sal(2,n) \\ Sal(1,n) \end{matrix}$$

und

$$[H_{cs}(3)] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \end{bmatrix} \begin{matrix} Wal_w(0,n) \\ Cal(1,n) \\ Cal(2,n) \\ Cal(3,n) \\ Sal(4,n) \\ Sal(3,n) \\ Sal(2,n) \\ Sal(1,n) \end{matrix}.$$

Die Elemente von $[H_{cs}(L)]$, $(L = 1,2,\ldots,r)$ können direkt aus folgendem Ausdruck erzeugt werden

$$h_{cs}(k,n) = (-1)^{\sum_{j=0}^{r-1} P_j V_j}, \qquad k, n = 0,1,\ldots,N-1,$$

wobei $P_0 = U_{r-1} + U_{r-2}$, $P_1 = U_{r-2} + U_{r-3}$, $\ldots$ $P_{r-2} = U_1 + U_0$, $P_{r-1} = U_0$ und $(n)_{dez} = (V_{r-1}V_{r-2}V_{r-3}\ldots V_1V_0)_{bin}$ (vgl.(2.5)) gelten.
In Matrixnotation lauten die $(WHT)_{cs}$ und ihre Inverse $(IWHT)_{cs}$:

$$\underline{X}_{cs} = [H_{cs}(L)]\,\underline{x}, \qquad \underline{x} = N^{-1}\,[H_{cs}(L)]\,\underline{X}_{cs}, \qquad L = 1,2,\ldots,r,$$

Für die entsprechende Summendarstellung gilt

$$X_{cs}(k) = \sum_{n=0}^{N-1} x(n)\, h_{cs}(k,n) \quad \longleftrightarrow \quad x(n) = N^{-1} \sum_{k=0}^{N-1} X_{cs}(k)\, h_{cs}(k,n),$$

$$k = 0,1,\ldots,N-1, \qquad n = 0,1,\ldots,N-1.$$

2.5 Haar-Transformation

Die Haar-Funktionen sind ein vollständiges System von orthonormalen rechteckigen Basisfunktionen. Sie wurden zuerst in einer Arbeit des ungarischen Mathematikers A. Haar im Jahre 1910 beschrieben.

Den Funktionensystemen mit rechteckigen Basisfunktionen gilt heute ein großes Interesse, da sie rechnerisch einfacher sind als die traditionelle Sinus/Cosinus-Basismenge. Die Haar-Basisfunktionen sind speziell interessant, weil sie die charakteristische Eigenschaft haben, daß sie sowohl globale als auch lokale Funktionseigenschaften besitzen.

Die Haar-Funktionen sind wie folgt definiert [2.15]:

$$HAR(0,n/N) = 1 \qquad \text{für } 0 \leqq n \leqq N,$$

$$HAR(1,n/N) = \begin{cases} 1 & \text{für } 0 \leqq n < (N/2), \\ -1 & \text{für } (N/2) \leqq n \leqq N, \end{cases}$$

$$HAR(2,n/N) = \begin{cases} \sqrt{2} & \text{für } 0 \leqq n < (N/4), \\ -\sqrt{2} & \text{für } (N/4) \leqq n < (N/2), \\ 0 & \text{für } (N/2) \leqq n \leqq N, \end{cases}$$

$$HAR(3,n/N) = \begin{cases} 0 & \text{für } 0 \leqq n < (N/2), \\ \sqrt{2} & \text{für } (N/2) \leqq n < (3N/4), \\ -\sqrt{2} & \text{für } (3N/4) \leqq n \leqq N, \end{cases}$$

.

.

$$HAR(2^p+m,n/N) = \begin{cases} (\sqrt{2})^p & \text{für } m(N/2^p) \leqq n < (m+(1/2))(N/2^p), \\ -(\sqrt{2})^p & \text{für } (m+(1/2))(N/2^p) \leqq n < (m+1)(N/2^p), \\ 0 & \text{sonst} \end{cases}$$

mit $p = 0,1,\ldots,$ und $m = 0,1,\ldots,2^p-1$.

Die Haar-Funktionen sind auf dem abgeschlossenen Intervall [0,1] definiert, können aber außerhalb des Intervalls periodisch fortgesetzt werden. Die Basisfunktionen werden in geordnete Untermengen eingeteilt, die Stufen oder Familien genannt werden. Innerhalb jeder Untermenge gibt es eine weitere Einteilung, die Ordnung oder Termanzahl der Funktion heißt. Die m-te Stufe enthält 2^m-mal soviele Terme wie die (m-1)-te Stufe. Jeder Term in einer Stufe ist eine verschobene Version aller anderen Terme in dieser Stufe. Die Basisfunktionen für die ersten drei Stufen sind im Bild 2.8 dargestellt.

Man beachte, daß der erste Term jeder Stufe ähnlich dem ersten Term der vorhergehenden Stufe ist, jedoch ist er auf die Hälfte des Intervalls zusammengedrängt. Wegen der rechteckigen Natur der Funktionen gibt es Diskontinuitäten. Die Funktionswerte an diesen Stellen werden deswegen als Mittelwert der Werte auf beiden Seiten der Funktionsdiskontinuität definiert.

Der Haar-Transformation liegt die Basis HAR(k,n/N), d.h. die diskreten Haar-Funktionen zugrunde, die periodisch, orthogonal, und vollständig sind (vgl. Bild 2.8). Die Haar-Transformation und ihre Inverse sind wie folgt definiert:

$$X_{HA}(k) = \sum_{n=0}^{N-1} x(n) \cdot HAR(k,n/N), \qquad k = 0,1,\ldots,N-1,$$

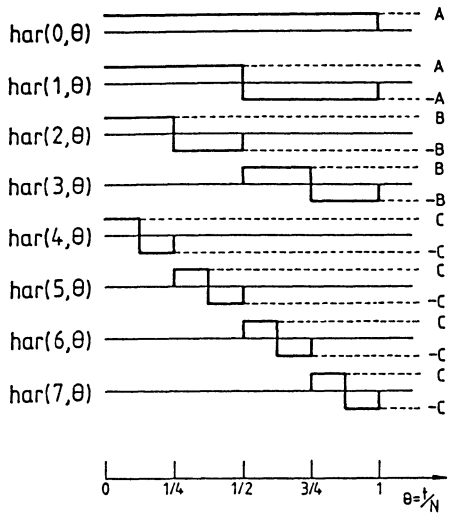


Bild 2.8. Haar-Basisfunktionen $har(k,\theta)$ für N=8 mit A=1, $B=\sqrt{2}$ und C=2, nach [1.1]

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X_{HA}(k) \cdot HAR^{-1}(k,n/N), \qquad n = 0,1,\ldots,N-1,$$

und $N = 2^r$.

In Matrixschreibweise haben wir

$$\underline{X}_{HA} = [HAR(L)]\,\underline{x} \qquad \text{und} \qquad \underline{x} = N^{-1}\,[HAR(L)]^{-1}\underline{X}_{HA}$$

mit $[HAR(L)]^{-1} = [HAR(L)]$ und $L = 1,2,\ldots,r$.

Die (N×N)-Haar-Matrix [HAR(L)] ist orthogonal, d.h.

$$[HAR(L)][HAR(L)]^T = N[I(L)].$$

Haar-Matrizen der Ordnung r=1 bis r=3 sind nachfolgend angegeben:

$$[HAR(1)] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

$$[HAR(2)] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ & & & \\ \sqrt{2}\ (1 & -1 & 0 & 0) \\ \sqrt{2}\ (0 & 0 & 1 & -1) \end{bmatrix},$$

$$[HAR(3)] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2}\ (1 & 1 & -1 & -1 & 0 & 0 & 0 & 0) \\ \sqrt{2}\ (0 & 0 & 0 & 0 & 1 & 1 & -1 & -1) \\ & & & & & & & \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{bmatrix}.$$

Haar-Matrizen sind zwar orthogonal, aber nicht symmetrisch. Im Unterschied zu den Walsh-Hadamard-Matrizen sind sie keine Kronecker-Matrizen, können aber durch eine Summe von Kronecker-Matrizen dargestellt werden (vgl. Abschnitt 3.3). Während bei der DFT und bei der WHT jeder Transformationskoeffizient eine Funktion von allen Koordinaten im ursprünglichen Werteraum ist, d.h. die Basisfunktionen außer an den endlich vielen Nulldurchgängen ungleich null sind, trifft dies bei der diskreten Haar-Transformation nur für die ersten beiden Koeffizienten zu. Somit sind die höheren Haar-Koeffizienten lokaler Art.

2.6 Slant-Transformation

Auf der Grundlage der DFT, der WHT und der Haar-Transformation lassen sich eine ganze Reihe anderer orthogonaler Transformationen konstruieren. Die diskrete Slant-Transformation (DSLT) wurde von Enomoto und Shibata [2.6] zum ersten Mal zur Bandbreitenkompression in einem Fernsehübertragungs-System benutzt. Obwohl die Bedeutung dieser Transformation durch die Entwicklung der Cosinus-Transformation abgenommen hat, spielt sie noch eine wichtige Rolle, wenn eine Beziehung unter verschiedenen orthogonalen Transformationen gesucht wird.

[SL(L)] bezeichnet die Slant-Matrix der Größe N×N mit $N=2^r$. Für N=2 (r=1) gilt

$$[SL(1)] = (1/\sqrt{2}) \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Die Slant-Matrix erster Ordnung [SL(1)] entspricht also (wie alle Orthogonalmatrizen) der Hadamard-Matrix erster Ordnung $[H_H(1)]$. Die Slant-Transformationsmatrizen können rekursiv als Produkte von schwach besetzten Matrizen (engl.: sparse matrices) ermittelt werden, z.B:

$$[SL(2)] = (1/\sqrt{2}) \begin{bmatrix} 1 & | & | & | \\ - & - & - & - \\ | & b_4 & a_4 & | \\ | & | & | & | \\ | & a_4 & -b_4 & | \\ - & - & - & - \\ | & | & | & 1 \end{bmatrix} \begin{bmatrix} [SL(1)] & [SL(1)] \\ - & - \\ [SL(1)] & -[SL(1)] \end{bmatrix}$$

$$= (1/2) \begin{bmatrix} 1 & 1 & 1 & 1 \\ b_4+a_4 & -b_4+a_4 & b_4-a_4 & -b_4-a_4 \\ -b_4+a_4 & -b_4-a_4 & b_4+a_4 & b_4-a_4 \\ 1 & -1 & -1 & 1 \end{bmatrix}.$$

Dabei werden die reellen Konstanten b_4 und a_4 mit Hilfe der folgenden Bedingungen bestimmt:

a) Die Abstände zwischen jeweils zwei Elementen müssen gleich sein, d.h. der Abstand zwischen den rsten beiden Elementen des Slant-Vektors [SL(2)] (vgl. die zweite Zeile von [SL(2)])

$$(b_4 + a_4) - (-b_4 + a_4) = 2b_4$$

muß gleich dem Abstand zwischen dem zweiten und dem dritten Element sein:

$$(-b_4 + a_4) - (b_4 - a_4) = 2a_4 - 2b_4.$$

Daraus folgt $2b_4 = 2a_4 - 2b_4$ und deshalb $a_4 = 2b_4$.

b) [SL(2)] muß orthogonal sein, d.h.

$[SL(2)][SL(2)]^T = [I(2)]$:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ b_4+a_4 & -b_4+a_4 & b_4-a_4 & -b_4-a_4 \\ -b_4+a_4 & -b_4-a_4 & b_4+a_4 & b_4-a_4 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & b_4+a_4 & -b_4+a_4 & 1 \\ 1 & -b_4+a_4 & -b_4-a_4 & -1 \\ 1 & b_4+a_4 & b_4+a_4 & -1 \\ 1 & -b_4-a_4 & b_4-a_4 & 1 \end{bmatrix} = 4 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Die Orthonormalität zeigt sich, indem

$$(1/4)\ [\ (b_4+a_4)^2 + (-b_4+a_4)^2 + (b_4-a_4)^2 + (-b_4-a_4)^2\ ] = 1$$

mit $a_4 = 2b_4$ wird.

So erhält man nach einigen Umformungen $b_4 = 1/\sqrt{5}$ und $a_4 = 2/\sqrt{5}$ und damit

$$[SL(2)] = (1/2) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1/\sqrt{5}\ (3 & 1 & -1 & -3) \\ 1/\sqrt{5}\ (1 & -3 & 3 & -1) \\ 1 & -1 & -1 & 1 \end{bmatrix}.$$

In analoger Weise erzeugt man die Slant-Matrix [SL(L+1)] durch Zusammensetzung aus [SL(L)]:

$$[SL(L+1)] = (1/\sqrt{5}) \begin{bmatrix} [I(L-1)] & & & \\ \hline & b(L+1) & a(L+1) & \\ & 1 & & \\ & . & & \\ & . & & \\ & 1 & & \\ \hline & a(L+1) & -b(L+1) & \\ & & 1 & \\ & & . & \\ & & . & \\ & & 1 & \\ \hline & & & [I(L-1)] \end{bmatrix}$$

$$\times \begin{bmatrix} [SL(L)] & [SL(L)] \\ \hline [SL(L)] & -[SL(L)] \end{bmatrix}.$$

wobei $[I(L-1)]$ die Einheitsmatrix der Ordnung L−1 ist. a(L+1), b(L+1) sind Konstanten, die wie folgt berechnet werden:

$$a(L+1) = a_{2N} = [3N^2/(4N^2-1)]^{1/2}.$$

$$b(L+1) = b_{2N} = [(N^2-1)/(4N^2-1)]^{1/2}.$$

Wir benutzen die allgemeine Slant-Matrix [SL(L)] zur Definition der Slant-Transformation und ihrer Inversen:

$$\underline{X}_{SL} = [SL(L)]\,\underline{x} \qquad \underline{x} = [SL(L)]^T\,\underline{X}_{SL} \qquad \text{mit} \qquad L = 1,2,\ldots,r,$$

wobei wegen der Orthogonalität $[SL(L)]^T = [SL(L)]^{-1}$ ist.

$$\underline{X}_{SL}^T = \{X_{SL}(0), X_{SL}(1), \ldots, X_{SL}(N-1)\}$$

ist der Slant-Koeffizienten-Vektor, $\underline{x}^T = \{x(0),x(1),\ldots,x(N-1)\}$ der Daten-

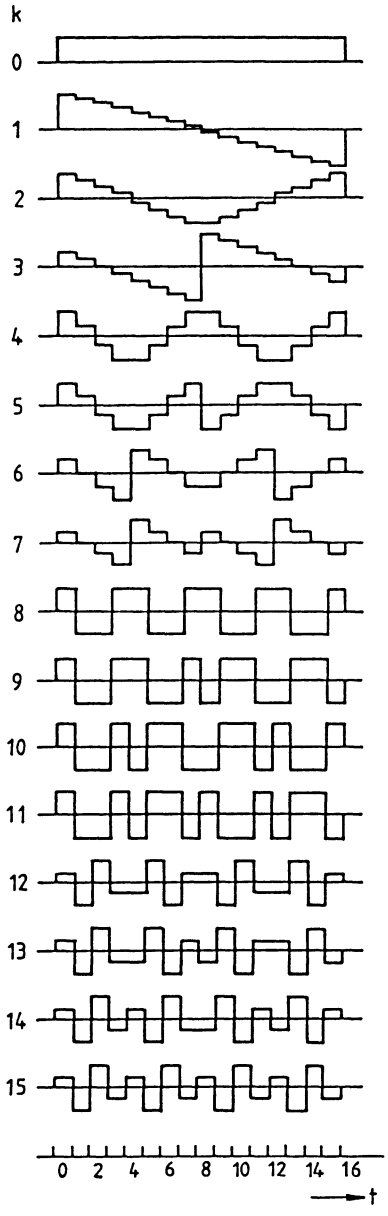


Bild 2.9. Slant-Basisfunktionen sla(k,t) für N=16, nach [2.17]

vektor und [SL(L)] die (N×N)-Slant-Matrix. Die Basisvektoren der Slant-Transformation sind die Zeilen der Slant-Matrix [SL(L)] [2.17]. Bild 2.9 zeigt die ersten 16 (kontinuierlichen) Slant-Funktionen.

Die Beziehung zwischen den Walsh-Funktionen $wal_w(k,t)$ und den Slant-Funktionen sla(k,t) gestaltet sich für N=8 wie folgt [2.15]:

$$sla(0,t) = wal_w(0,t),$$

$$sla(1,t) = (1/\sqrt{21})[4 \cdot wal_w(1,t) + 2 \cdot wal_w(3,t) + wal_w(7,t)],$$

$$sla(2,t) = (1/\sqrt{5})\ [2 \cdot wal_w(1,t) + wal_w(6,t)],$$

$$sla(3,t) = (1/\sqrt{21})(1/\sqrt{5})\ [-5 \cdot wal_w(1,t) + 8 \cdot wal_w(7,t)],$$

$$sla(4,t) = wal_w(4,t),$$

$$sla(5,t) = wal_w(5,t),$$

$$sla(6,t) = (1/\sqrt{5})\ [-wal_w(2,t) + 2 \cdot wal_w(6,t)],$$

$$sla(7,t) = (1/\sqrt{5})\ [-wal_w(3,t) + 2 \cdot wal_w(7,t)].$$

3 Strategien, Berechnungs- und Darstellungsformen

3.1 Berechnungsstrategien

3.1.1 Direkte Berechnung der Transformationskoeffizienten

Es sei die Aufgabe gestellt, die Koeffizienten einer diskreten Orthogonaltransformation zu berechnen. Zur Durchführung dieser Aufgabe kann man sich verschiedener Strategien bedienen. Bei der *direkten* Berechnungsmethode werden die Koeffizienten direkt aus den Objektdaten bestimmt, ohne daß hierbei irgendwelche sinnvollen Zwischenergebnisse erhalten werden: Man ist lediglich am vollständigen oder unvollständigen Satz der Koeffizienten interessiert. Im allgemeinen hängt jeder Koeffizienten X(k) von allen Objektelementen x(n) ab (Ausnahme: Haar-Transformation):

$$X(k) = \sum_{n=1}^{N} x(n) g_k(n)$$

Dabei sind die g_k, $k = 1,2,\ldots,N$, die Basisfunktionen der Transformation. Wie man sieht, sind hierfür etwa N^2 Operationen (N^2 Multiplikationen plus N^2-N Additionen) notwendig. Dieser Rechenaufwand reduziert sich entsprechend, wenn nicht alle Koeffizienten berechnet werden sollen. Falls es gelingt, die Rechenvorschrift als Radix-2-Algorithmus darzustellen, benötigt man jedoch nur $N \log_2 N$ Operationen, um sämtliche N Koeffizienten zu berechnen (vgl. Abschnitt 3.4).

3.1.2 Berechnung aus Koeffizienten von Teilfolgen

Andere Strategien der Koeffizientenberechnung bestehen darin, diese nicht direkt aus den Daten des Objektbereichs, sondern aus anderen Koeffizienten

zu berechnen. Hierfür kommen entweder die Koeffizienten niedrigerer Ordnung der gleichen Transformation oder die Koeffizienten gleicher Ordnung einer anderen Transformation in Frage. Motivation für die Anwendung einer solchen Strategie ist stets die Verringerung des Rechenaufwandes. Es ist leicht einzusehen, daß man an Operationen spart, falls es gelingt, Transformationskoeffizienten aus vorhandenen Koeffizienten niedigerer Ordnung zu ermitteln. Dabei verstehen wir unter Koeffizienten niedrigerer Ordnung solche, die aus Bruchteilen der Datenfolge berechnet wurden, z.B. aus zwei Hälften, vier Vierteln, etc. der Objektdaten. Allerdings ist es nicht allgemein möglich, Zerlegungen der Datenfolge anzugeben, so daß Koeffizienten höherer Ordnung aus ihnen gewonnen werden können. Sofern die Koeffizienten jeweils aus disjunkten Teilmengen von Koeffizienten niedrigerer Ordnung gewonnen werden können, sprechen wir von einer hierarchischen Berechnung. In den Abschnitten 5.3 und 5.4 wird auf diese Berechnungsstrategie näher eingegangen.

3.1.3 Berechnung aus den Koeffizienten einer anderen Transformation

Weniger einleuchtend ist, daß es vorteilhaft sein kann, aus den Daten zunächst die Koeffizienten einer anderen Transformation zu berechnen und aus diesen diejenigen der gewünschten Transformation. Dieses Verfahren bietet sich natürlich dann an, wenn die Koeffizienten beider Transformationen benötigt werden. Es hat sich jedoch gezeigt, daß unter gewissen Umständen derartige "Umwege" bei der Berechnung der Koeffizienten nur einer Transformation zweckmäßig sein können, siehe z.B. [3.1], [3.2].

Wir zeigen hier allgemein, wie man die Koeffizienten X_f einer Basis $\{f_k\}$ aus den Koeffizienten X_g einer Basis $\{g_k\}$ ermitteln kann. Seien $f_k(n)$ und $g_k(n)$, $k = 1,2,\ldots,N$, die Basisfunktionen der "F"-Transformation und der "G"-Transformation. Die zugehörigen Transformationskoeffizienten in Matrixnotation sind dann

$$\underline{X}_f = [F]\cdot\underline{x}, \quad \text{bzw.} \quad \underline{X}_g = [G]\cdot\underline{x}.$$

Mit den inversen Ausdrücken

$$\underline{x} = N^{-1}\ [F]^{-1}\underline{X}_f = N^{-1}\ [G]^{-1}\underline{X}_g$$

ergibt sich

$$\underline{X}_f = N^{-1}[F][G]^{-1}\underline{X}_g = N^{-1}[T]\ \underline{X}_g .$$

Die Transformationsmatrix

$$[T] = [F][G]^{-1}$$

ist als regulär vorauszusetzen, was im Fall von orthogonalen Transformationen gegeben ist.

Anwendungen auf ein- und zweidimensionale Transformationen werden in den Abschnitten 4.2.6 und 5.4 diskutiert.

3.2 Anwendung der Restklassenarithmetik

Die Durchführung einer diskreten Transformation ist im allgemeinen eine rechenintensive Aufgabe. Man bemüht sich deshalb Rechenverfahren anzuwenden, die eine Reduktion des Rechenaufwandes erlauben. Hierzu gehören einerseits die Zerlegung des Problems in Teilprobleme und die optimale Verwendung von Zwischenergebnissen, wie sie bei den Radix-p-Algorithmen und bei den Primfaktor-Algorithmen angewandt werden. Andererseits profitieren letztere auch von der Anwendung der Restklassenarithmetik. Da diese weniger bekannt ist, wird hier ein kurzer Abriß ohne Anspruch auf Vollständigkeit gegeben.

Die Vorteile der Restklassenverfahren bestehen allgemein darin, daß keine Überträge zu verarbeiten sind. Bei der Anwendung auf diskrete Transformationen lassen sich ähnlich große Einsparungen an Rechenoperationen erzielen wie durch Verwendung der Radix-p-Algorithmen. Die Restklassenarithmetik setzt die Anwendung des "Chinesischen Rest-Theorems" (CRT) voraus, das nachfolgend erläutert wird.

Im ersten Jahrhundert A.D. schrieb Sun-Tzu in China ein Arithmetikbuch, das einen eingebetteten Vers enthält. Dieser Tai-Yen (größte Verallgemeinerung) genannte Vers bestimmt eine Zahl, die bei Division durch 3, 5 und 7 die Re-

今有物不知其數三三數之賸二五五數之賸
三七七數之賸二問物幾何
答曰二十三
術曰三三數之賸二置一百四十五五數
之賸三置六十三七七數之賸二置三十
并之得二百三十三以二百一十減之即
得凡三三數之賸一則置七十五五五數之
賸一則置二十一七七數之賸一則置十
五一百六以上以一百五減之即得

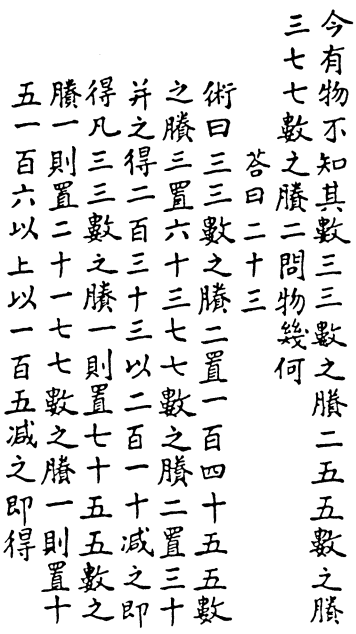


Bild 3.1. Ursprünglicher Text des Chinesischen Rest-Theorems

ste 2, 3 und 2 ergibt. Mit diesem Vers hatte Sun-Tzu ein Restklassensystem mit den Prim-Moduli $\{3,5,7\}$ beschrieben. Bild 3.1 zeigt eine Reproduktion des ursprünglichen chinesischen Textes [3.3]. Diese Regel wurde von anderen Wissenschaftlern weiter verfeinert und später als Chinesisches Rest-Theorem (CRT) bezeichnet.

Wir betrachten zwei ganze Zahlen A_0 und A_1. Mit Hilfe des Euklidschen Divisionsalgorithmus' bestimmen wir ihren größten gemeinsamen Teiler:

$$
\begin{aligned}
A_0 &= L_1 A_1 + A_2 && A_2 < A_1 \\
A_1 &= L_2 A_2 + A_3 && A_3 < A_2 \\
&\vdots \\
A_{i-1} &= L_i A_i + A_{i+1} && A_{i+1} < A_i \\
&\vdots \\
A_{N-1} &= L_N A_N .
\end{aligned}
$$

Dabei sind die L_i ganze Zahlen. Der größte gemeinsame Teiler von A_0 und A_1 ist dann A_N:

$$GGT(A_0, A_1) = A_N.$$

Seien b_i, $i=1,2,3,\ldots,m$, Primzahlen oder paarweise teilerfremde (relative prime) Zahlen. Die Menge der linearen Kongruenzen

$$A = a_i \text{ modulo } b_i, \qquad a_i \in N$$

hat eine einzige Lösung modulo B mit

$$B = \prod_{i=1}^{m} b_i .$$

Wenn die b_i Primzahlen sind, gilt

$$GGT(b_i, (B/b_i)P_i) \equiv 1, \qquad \text{d.h. } (B/b_i)P_i \equiv 1 \text{ modulo } b_i . \tag{3.1}$$

Gleichung (3.1) hat dann eine eineindeutige Lösung P_i, die durch den Euklidschen Algorithmus berechnet werden kann.

Wir betrachten nun die Beziehung

$$A = \sum_{i=1}^{m} (B/b_i)a_i P_i \text{ modulo } B. \tag{3.2}$$

Weil $(B/b_v)a_v P_v$ modulo $b_u \equiv 0$ für $v \neq u$ und $u,v \in \{1,2,3,\ldots,m\}$, erhalten wir dann aus (3.1) und (3.2)

$$A \equiv \{(B/b_u)a_u P_u \text{ modulo } b_u\} \equiv \{a_u \text{ modulo } b_u\}.$$

Ein einfaches Beispiel für die Anwendung des CRT ist es, die folgenden simultanen Kongruenzen zu lösen:

Seien $\langle A \rangle_7 = 5$, $\langle A \rangle_{11} = 7$, $\langle A \rangle_{13} = 11$, $\langle A \rangle_5 = 3$, $\langle A \rangle_3 = 1$. Die Schreibweise $\langle A \rangle_{b_i} = a_i$ bedeutet dabei A mod $b_i \equiv a_i$. Die Zahl A ist zu bestimmen. Dann ist

$a_1 = 5$, $a_2 = 7$, $a_3 = 11$, $a_4 = 3$, $a_5 = 1$, $b_1 = 7$, $b_2 = 11$, $b_3 = 13$, $b_4 = 5$, $b_5 = 3$

und

$$B = \prod_{i=1}^{5} b_i = 15015.$$

Mit (3.1) erhalten wir

$$\{(B/b_1)P_1\} = \{b_2b_3b_4b_5P_1\} \equiv 1 \text{ modulo } b_1,$$

$$<2145\ P_1>_7 = 1,\ P_1 = 5,$$

$$\{(B/b_2)P_2\} = \{b_1b_3b_4b_5P_2\} \equiv 1 \text{ modulo } b_2,$$

$$<1365\ P_2>_{11} = 1,\ P_2 = 1,$$

$$\{(B/b_3)P_3\} = \{b_1b_2b_4b_5P_3\} \equiv 1 \text{ modulo } b_3,$$

$$<1155\ P_3>_{13} = 1,\ P_3 = 6,$$

$$\{(B/b_4)P_4\} = \{b_1b_2b_3b_5P_4\} \equiv 1 \text{ modulo } b_4,$$

$$<3003\ P_4>_5 = 1,\ P_4 = 2,$$

$$\{(B/b_5)P_5\} = \{b_1b_2b_3b_4P_5\} \equiv 1 \text{ modulo } b_5,$$

$$<5005\ P_5>_3 = 1,\ P_5 = 1.$$

Dann haben wir mit (3.2)

$$A = \{\sum_{i=1}^{5} (B/b_i)a_iP_i\} \text{ modulo } B$$

$$\equiv 162433 \text{ modulo } 15015 = 12283.$$

Nun prüfen wir, ob diese Lösung die Bedingungen erfüllt:

$$<12283>_7 = 5,\ <12283>_{11} = 7,\ <12283>_{13} = 11,\ <12283>_5 = 3,\ <12283>_3 = 1.$$

Das Chinesische Rest-Theorem kann zur Definition von Restklassensystemen (RNS) benutzt werden, die eine schnelle Durchführung von arithmetischen Operationen ohne Übertrag von Stelle zu Stelle ermöglichen. In einem solchen System wird eine ganze Zahl Z durch ihre Reste Z_i modulo einer Menge von relativen Primzahlen Y_i repräsentiert. Die Addition bzw. Multiplikation von zwei ganzen Zahlen Z und Y wird durch getrenntes Addieren bzw. Multiplizie-

ren ihrer jeweiligen Reste Z_i und Y_i (modulo P_i) ohne Überträge von einem Rest auf einen anderen durchgeführt [2.2].

Im RNS werden also ganze Zahlen mit Bezug auf die Menge der Prim-Moduli P_i dargestellt:

$$P = \{ P_1, P_2, P_3, \ldots, P_m \}$$

$$GGT(P_i, P_j) = 1, \qquad i,j \in \{1,2,\ldots,m\}, \qquad i \neq j.$$

Jede ganze Zahl $Z_L \in \{0,1,2,\ldots,L-1\}$, $L = P_1 P_2 P_3 \ldots P_m$, hat eine eindeutige RNS-Repräsentation:

$$Y \longrightarrow (Y_1, Y_2, Y_3, \ldots, Y_m), \quad Y_i = Y \text{ modulo } P_i, \quad Y \in Z_L.$$

Symbolisiert das Zeichen "o" die Operationen +, − und ×, dann genügt die Operation $Z = Y \circ Q$ der Beziehung

$$Z \longrightarrow (Z_1, Z_2, Z_3, \ldots, Z_L), \quad Z_i = Y_i \circ Q_i \text{ modulo } P_i.$$

Aufgrund dieser Tatsache können die Operationen $Y_i \circ Q_i$ in Tabellen abgelegt werden und z.B. durch schnelle Halbleiterspeicher realisiert werden. Für die Hardware-Realisierung einer FFT-Struktur, die ein Array von ROMs (Read Only Memories) benutzt, kann das RNS vorteilhaft angewandt werden [3.4]. Die Arithmetikoperationen basieren dabei völlig auf dem RNS.

Das RNS und das CRT werden häufig für die numerische Transformationsberechnung verwendet. Zum Beispiel ist das CRT das Fundament der Index-Bezeichnung beim Primfaktor-Algorithmus. Weil die Berechnungen sowohl für Addition als auch für Multiplikation unabhängig voneinander in der Restklassenarithmetik durchgeführt werden können, läßt sich eine Funktion in der Restklasse durch Ausführung der funktionell gleichen Operationen auf dem Restcode berechnen. Das RNS findet deshalb wegen seiner Fähigkeit zur Unterstützung sehr schneller Arithmetik immer mehr Beachtung. Dabei wird von der Tatsache Gebrauch gemacht, daß die Restklassensysteme das Rechnen mit großen Integer-Zahlen durch Operationen an den kleineren Resten ersetzen.

3.3 Faktorisierung der Transformationsmatrix

Für die Lösung der in diesem Buch behandelten Probleme ist es nützlich, das Rechnen mit Matrizen höherer Ordnung auf die Berechnung von Matrizen niedrigerer Ordnungen zurückzuführen. Dies kann mit Hilfe einer Zerlegung der gegebenen Matrizen in sogenannte Blöcke (auch Untermatrizen genannt) verwirklicht werden (vgl. Abschnitt 3.1.2). Man kann nämlich jede Matrix auf vielfache Weise aus Matrizen niedrigerer Ordnung zusammensetzen. Außer einer solchen Zerlegung der Transformationsmatrix macht man für die Transformationsalgorithmen fast regelmäßig von der Faktorisierung der Matrizen Gebrauch. Die Faktorisierung ermöglicht die Reduktion einer umfangreichen Menge von Operationen auf weniger und voneinander unabhängige Operationen.

In diesem Abschnitt werden zunächst einige wichtige Begriffe der Matrizenrechnung dargestellt, die man für eine geschlossene Behandlung der Faktorisierung benötigt. Die Begründung und Entwicklung der Matrixfaktorisierung stützt sich auf die Definition der Matrizen und ihre Rechenregeln.

Eine Gesamtheit von Zahlen, die in Form eines rechteckigen Schemas von n Zeilen und m Spalten angeordnet sind, heißt eine rechteckige Matrix. Wenn die rechteckige Matrix die gleiche Anzahl von Zeilen und Spalten enthält (d.h. n=m), nennen wir diese Matrix eine quadratische Matrix. In diesem Fall heißt n die Ordnung der Matrix. Ein Sonderfall der quadratischen Matrix ist die symmetrische Matrix. Eine symmetrische Matrix [A] ist gleich ihrer Transponierten $[A]^T = [A]$. Eine quadratische Matrix [A] heißt hermitisch wenn $[A] = [A]^*$ gilt. Die Elemente in der Hauptdiagonalen einer hermitischen Matrix sind reell.

Eine wichtige Rolle spielen die Diagonalmatrizen, die ein Sonderfall einer symmetrischen Matrix sind. Bei diesen Matrizen sind nur die in der Hauptdiagonale stehenden Elemente von Null verschieden. Sind in einer Diagonalmatrix [A] alle Elemente identisch, d.h. $a_{ii} = a$, dann spricht man von einer Skalarmatrix. Ein Sonderfall einer Skalarmatrix ist die Einheitsmatrix $[I_N]$ bei der die $a_{ii} = 1$ sind. Darüber hinaus wird eine Matrix, deren Elemente entweder 1 oder 0 sind, als Binärmatrix bezeichnet. Ein Sonderfall einer Binärmatrix ist die Permutationsmatrix, bei der in jeder Zeile und jeder Spalte nur eine 1 enthalten ist.

Im übrigen nennt man eine quadratische Matrix, bei der längs der Hauptdiagonalen quadratische Blöcke angeordnet sind, alle übrigen Elemente jedoch Null sind, eine Quasidiagonalmatrix. Beispiel einer quasidiagonalen Matrix 7-ter Ordnung:

$$\left[\begin{array}{cc|ccc|cc} a_{11} & a_{12} & 0 & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & b_{11} & b_{12} & b_{13} & 0 & 0 \\ 0 & 0 & b_{21} & b_{22} & b_{23} & 0 & 0 \\ 0 & 0 & b_{31} & b_{32} & b_{33} & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & c_{11} & c_{12} \\ 0 & 0 & 0 & 0 & 0 & c_{21} & c_{22} \end{array}\right]$$

Sofern eine Transformationsmatrix eine entsprechend regelmäßige Struktur besitzt, kann sie in ein Kronecker-Produkt zerlegt oder in der Form eines Produktes von schwach besetzten Matrizen geschrieben werden.

Ein Kronecker-Produkt oder "direktes Produkt" zweier Matrizen wird durch das Zeichen "⊗" symbolisiert und als eine parzellierte Matrix definiert. Seien $[A]=[a_{ij}]$ eine Matrix der Ordnung m×n und [B] eine andere Matrix der Ordnung r×s. Das Kronecker-Produkt von [A] und [B] wird definiert als

$$[A]\otimes[B] = \begin{bmatrix} a_{11}[B] & a_{12}[B] & \dots & a_{1n}[B] \\ . & . & \dots & . \\ . & . & \dots & . \\ . & . & \dots & . \\ a_{m1}[B] & a_{m2}[B] & \dots & a_{mn}[B] \end{bmatrix}$$

[A]⊗[B] ist eine Matrix der Ordnung mr×ns. Diese Matrix hat m×n Blöcke. Der (ij)te-Block ist die Matrix $a_{ij}[B]$ der Ordnung (r×s). Für die Kronecker-Produktbildung gelten die folgenden Rechenregeln, die ohne Beweis angegeben werden [3.5], [2.15]:

a) $([A]+[B])\otimes[C] = [A]\otimes[C] + [B]\otimes[C]$,

b) $([A]\otimes[B])([C]\otimes[D]) = ([A][C])\otimes([B][D])$,

c) $([A]\otimes[B])^{-1} = [A]^{-1}\otimes[B]^{-1}$, $\det[A]\neq 0$, $\det[B]\neq 0$,

d) $([A]\otimes[B])^{T} = [A]^{T}\otimes[B]^{T}$.

e) Wenn $m=n$ und $r=s$, gilt $\det([A]\otimes[B]) = (\det[A])^{n}(\det[B])^{s}$.

f) Wenn $[G] = [A] \otimes [B]$, gilt $[G] = ([A]\otimes[I]_{s})([I]_{n}\otimes[B])$. (3.3)

Die Faktorisierung einer Transformationsmatrix in ein Produkt von schwach besetzten Quasidiagonalmatrizen wird am Beispiel der Walsh-Hadamard-Transformation gezeigt [3.6]. Für eine natürliche Zahl r sei $N=2^{r}$, $[H_{H}(L)]$ stellt die Hadamard-Matrix, und $[I(L)]$ die Einheitsmatrix mit den Dimensionen $N\times N$ dar ($1\leq L\leq r$). Die Elemente von $[H_{H}(L)]$ sind entweder +1 oder −1 und die Zeilen und Spalten sind paarweise orthogonal. Weil $[H_{H}(L)]^{-1} = N^{-1}[H_{H}(L)]$ gilt, sind die Vorwärts- und die Rücktransformationen eines N-dimensionalen Vektors gegeben durch

$$\underline{X} = [H_{H}(L)]\cdot\underline{x}, \qquad \underline{x} = N^{-1}[H_{H}(L)]\,\underline{X}, \qquad L = 1,2,\ldots,r.$$

Ein schneller Hadamard-Transformationsalgorithmus ist ein solcher, der den Vektor $\underline{X}$ in $(N\times K)$-Operationen $(K<N)$ berechnet.

Im Folgenden zeigen wir eine Faktorisierung von $[H_{H}(L)]$ unter Verwendung von $[H_{H}(1)]$, $[I(1)]=[I_{2}]$ und $[H(L-1)]$, die sich als Produkt von r schwach besetzten Matrizen (Sparse-Matrizen) ergibt. Der Algorithmus beruht auf folgender Rekursionsformel:

$$[H_{H}(L)] = [H_{H}(1)]\otimes[H_{H}(L-1)], \quad [H_{H}(1)] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{3.4}$$

Gemäß (3.3) gilt die folgende rekusive Faktorisierung (vgl. [3.6]):

$$[H_{H}(L)] = ([H_{H}(1)]\otimes [I(L-1)])([I(1)]\otimes[H_{H}(L-1)]). \tag{3.5}$$

Dabei sind L = 1,2,...,r und N = 2^r. Der Beweis von (3.5) ist einfach. Unter Verwendung der Rechenregeln für Kronecker-Produkte (3.3) haben wir:

$$([H_H(1)]\otimes[I(L-1)])([I(1)]\otimes[H_H(L-1)]) =$$

$$([H_H(1)][I(1)])\otimes([I(L-1)][H_H(L-1)]) = [H_H(1)]\otimes[H_H(L-1)] \qquad (3.6)$$

Gemäß (3.4) ist die rechte Seite von (3.6) genau $[H_H(L)]$. Damit ist der Beweis komplett.

Für N=4 gilt somit: $[H_H(2)] = ([H_H(1)] \otimes [I(1)])\ ([I(1)] \otimes [H_H(1)])$, d.h.

$$[H_H(2)] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ & & & \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ & & & \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ & & & \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}.$$

Die einfache rekursive Beziehung in (3.5) kann jetzt zur Formulierung einer Sparse-Matrixzerlegung von $[H_H(L)]$ verallgemeinert werden. Erweiterung des zweiten Terms von (3.5) mit aufeinanderfolgenden Hadamard-Matrizen niedrigerer Ordnung ergibt ein Produkt mit $r = \log_2 N$ Termen

$$[H_H(L)] = \prod_{i=1}^{r} \{\ [I(1)]\otimes[H_H(1)]\otimes[I(L-i)]\ \}. \qquad (3.7)$$

Jede dieser Matrizen in der Produktform ist nur schwach besetzt. Sie besitzen jeweils nur zwei von Null verschiedene Elemente in jeder Zeile und jeder Spalte. Deshalb benötigt die Transformation r·N Operationen. Zur Veranschaulichung betrachten wir die Matrix $[H_H(3)]$ in der Zerlegung von (3.7):

$$[H_H(3)] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ & & & & & & & \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 & 1 & 1 & -1 \end{bmatrix} =$$

$$
= \begin{bmatrix}
+ & 0 & 0 & 0 & + & 0 & 0 & 0 \\
0 & + & 0 & 0 & 0 & + & 0 & 0 \\
0 & 0 & + & 0 & 0 & 0 & + & 0 \\
0 & 0 & 0 & + & 0 & 0 & 0 & + \\
 & & & & & & & \\
+ & 0 & 0 & 0 & - & 0 & 0 & 0 \\
0 & + & 0 & 0 & 0 & - & 0 & 0 \\
0 & 0 & + & 0 & 0 & 0 & - & 0 \\
0 & 0 & 0 & + & 0 & 0 & 0 & -
\end{bmatrix}
\begin{bmatrix}
+ & 0 & + & 0 & & & & \\
0 & + & 0 & + & & 0 & & \\
+ & 0 & - & 0 & & & & \\
0 & + & 0 & - & & & & \\
 & & & & & & & \\
 & & & & + & 0 & + & 0 \\
 & & & & 0 & + & 0 & + \\
 & & 0 & & + & 0 & - & 0 \\
 & & & & 0 & + & 0 & -
\end{bmatrix}
\begin{bmatrix}
+ & + & 0 & 0 & & & & \\
+ & - & 0 & 0 & & 0 & & \\
0 & 0 & + & + & & & & \\
0 & 0 & + & + & & & & \\
 & & & & & & & \\
 & & & & + & + & 0 & 0 \\
 & & & & + & - & 0 & 0 \\
 & 0 & & & 0 & 0 & + & + \\
 & & & & 0 & 0 & + & -
\end{bmatrix}
$$

Dabei bezeichnen "+" bzw. "−" die Elemente 1 bzw. −1 in der Matrix.

In ähnlicher Weise kann man auch andere Transformationsmatrizen in ein Produkt von schwach besetzten Matrizen zerlegen, sofern ihre Struktur entsprechend regelmäßig ist. Bei der DFT müssen hierbei die Drehfaktoren berücksichtigt werden (vgl. Abschnitt 4.2.1).

Die Datenvektoren $\underline{x}$ und $\underline{X}$ des Objekt- und des Transformationsbereichs sind durch die Transformationsmatrix [T(L)] verknüpft

$$\underline{X} = [T(L)] \cdot \underline{x}.$$

Wir schreiben nun [T(L)] formal als Kronecker-Produkt von r Faktoren

$$[T(L)] = [T]_1 \otimes [T]_2 \otimes \ldots \otimes [T]_r, \tag{3.8}$$

wobei L = 1,2,....,r und $r = \log_2 N$ sind. Die Matrizen $[T]_1$, $[T]_2$,.... und $[T]_r$ sind von der Ordnung 2x2.

Falls in (3.8)

$$[T]_1 = [T]_2 = \ldots = [T]_r$$

gilt, ist [T(L)] eine Kronecker-Potenz,

$$[T(L)] = [T]_1 \otimes [T]_2 \otimes \ldots \otimes [T]_r = ([T]_1)^{\otimes r}$$

wobei "⊗r" bedeutet, daß r Matrizen $[T]_1$ durch ⊗ verbunden sind.

Nun betrachten wir noch einmal das Beispiel der Hadamard-Transformation. Aus (3.4) entwickeln wir

$$
\begin{aligned}
&[H_H(L)] = [H_H(1)] \otimes [H_H(L-1)] \\
&[H_H(L-1)] = [H_H(1)] \otimes [H_H(L-2)] \\
&[H_H(L-1)] = [H_H(1)] \otimes [H_H(L-3)] \\
&\quad . \\
&\quad . \\
&\quad . \\
&[H_H(2)] = [H_H(1)] \otimes [H_H(1)] \\
&[H_H(1)] = [H_H(1)] \otimes [H_H(0)] \\
&[H_H(0)] = 1
\end{aligned}
\qquad (3.9)
$$

Daraus folgt

$$[H_H(L)] = [H_H(1)] \otimes [H_H(1)] \otimes \ldots \otimes [H_H(1)] = ([H_H(1)])^{\otimes r},$$

worin der Faktor $[H_H(1)]$ genau r-mal enthalten ist.

Für N=8, r=3 ist zum Beispiel:

$$[H_H(3)] = [H_H(1)] \otimes [H_H(1)] \otimes [H_H(1)] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 & 1 & 1 & -1 \end{bmatrix}.$$

Falls sich die Transformationsmatrix durch eine Kronecker-*Potenz* darstellen läßt, kann die Transformation auch hierarchisch durchgeführt werden, d.h. die Koeffizienten können jeweils aus 2^r, r = 1,2,... Koeffizienten der

nächst niedrigeren Ordnung ermittelt werden (vgl. Abschnitt.5.4). Unter den orthogonalen Transformationen ist die $(WHT)_H$ die einzige, die diese Eigenschaft aufweist. Das erklärt einige ihrer interessanten Eigenschaften.

Um auch andere Transformationsmatrizen durch Kronecker-Produkte darstellen zu können, wurde von Fino und Algazi ein verallgemeinertes Kronecker-Produkt eingeführt [3.7]. Hierbei werden zwei Mengen $\{A\} = \{[A^0], [A^1], \ldots, [A^{m-1}]\}$ der Ordnung m und $\{B\} = \{[B^0], [B^1], \ldots, [B^{n-1}]\}$ der Ordnung n zu einer quadratischen Matrix [C] der Ordnung m×n verknüpft $[C] = \{A\} \circledast \{B\}$. Dabei ist das allgemeine Element

$$C_{ij} = C_{um+w,u'm+w} = A^{w}_{u,u'} \; B^{u'}_{w,w'}$$

mit

$$i = u \cdot m + w, \qquad u, u' = 0,1,\ldots,n-1$$
$$j = u' \cdot m + w', \qquad w, w' = 0,1,\ldots,m-1.$$

Für [C] erhält man

$$[C] = [P]^T [\mathrm{diag}\{A\}] \cdot [P] \cdot [\mathrm{diag}\{B\}].$$

Hierin bedeuten

$$\mathrm{diag}\,\{[M^1],\ldots,[M^{k-1}]\} = \begin{bmatrix} [M^1] & & & \\ & [M^2] & & \\ & & \ddots & \\ & & & [M^{k-1}] \end{bmatrix}$$

und [P] die "Perfect Shuffle"-Permutationsmatrix der Ordnung m×n. Die Bildung des verallgemeinerten Kronecker-Produkts ist im Bild 3.2 dargestellt. Für den Spezialfall, daß alle

$$[A^i]=[A], \qquad i = 1,2,\ldots,m-1$$

und alle

$$[B^j]=[B], \qquad j = 1,2,\ldots,n-1$$

geht (3.9) in das bekannte Kronecker-Produkt $[A] \circledast [B]$ über.

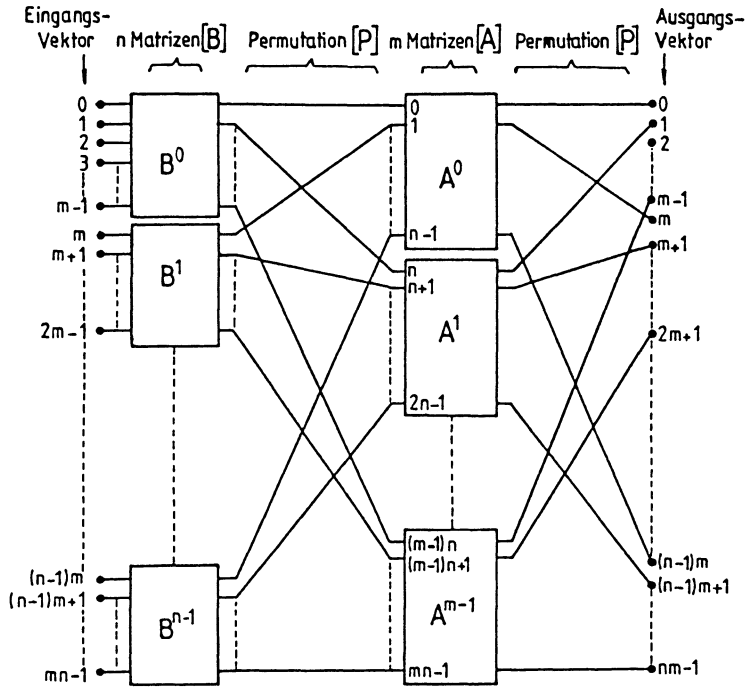


Bild 3.2. Zur Bildung des verallgemeinerten Kronecker-Produkts, nach [3.7]

Die Fourier-Transformationsmatrix (vgl. Abschnitt 4.2) der Ordnung $N = 2^r$, $r = 1,2,\ldots$, kann dann z.B. wie folgt dargestellt werden:

$$[F(L)] = [P]^T\{[F(L-1)]\otimes[F(1)]^T\}, \qquad 1 \leq L \leq r$$

$$[F(1)]^T = \begin{bmatrix} 1 & 1 \\ W^k & -W^k \end{bmatrix}, \qquad W = \exp(-j2\pi/N), \quad j = \sqrt{-1}, \quad k = 0,1,\ldots,N-1.$$

3.4 Indizierung der Daten im Objekt- und im Transformationsbereich

3.4.1 Aufteilung in Module

Ein "schneller" Algorithmus berechnet die N Transformationskoeffizienten aus den N Objektdaten mit weniger als $O(N^2)$ Operationen. Das wird möglich, indem

man die Daten in geeigneter Weise unterteilt und die Berechnung in einzelne *Module* aufteilt. Damit will man erreichen, daß Zwischenresultate mehrfach weiter verarbeitet werden und so die Anzahl der benötigten Operationen möglichst klein gehalten wird. Die Unterteilung der Datenmenge beruht auf einer geeigneten Indizierung.

Zu diesem Zweck zerlegt man die Anzahl N der diskreten Daten in Faktoren. N darf also keine Primzahl sein. Von den zahlreichen Möglichkeiten der Faktorisierung sind zwei von besonderem Interesse: Entweder wird N als ganzzahlige Potenz eines Radix $p \in \{2,3,\ldots\}$ gewählt, oder N ist das Produkt von m gegenseitig teilerfremden Faktoren $N_1, N_2, \ldots, N_m$. Damit ist die Anzahl der diskreten Datenwerte nur mit gewissen Einschränkungen frei wählbar.

Allgemein ist also $N = N_1 N_2 \ldots N_m$, wobei *entweder*

$$GGT(N_i, N_j) = 1, \qquad i,j = \{1,2,\ldots,m\} \qquad \text{und} \qquad i \neq j \tag{3.10}$$

gilt, *oder* $N_1 = N_2 = \ldots = N_r = p$ und damit

$$N = p^r \tag{3.11}$$

ist. Die Darstellung (3.11) von N durch eine ganzzahlige Potenz von p = 2,3... führt auf die Radix-p-Algorithmen.

Im ersten Fall (3.10) sind die Faktoren N_i, i = 1,2,...,m, gegenseitig teilerfremd (engl.: relative prime). Diese Faktorisierung ist Grundlage der Primfaktor-Algorithmen (PFA, vgl. die Abschnitte 4.2.3 und 4.3.3). Die Darstellung des Objekt-Index' n und des Transformations- (Frequenz-) Index' k gestaltet sich wie folgt. Zur Vereinfachung nehmen wir zunächst an, daß N nur in zwei Faktoren zerlegt wird

$$N = N_1 \cdot N_2, \qquad GGT(N_1, N_2) = 1.$$

Der Laufindex im Objektbereich $n = 0,1,2,\ldots,N-1$ wird nun durch zwei unabhängige Variable $n_1 = 0,1,2,\ldots,N_1-1$ und $n_2 = 0,1,2,\ldots,N_2-1$ ausgedrückt, derart daß

$$n = (C_1 n_1 + C_2 n_2) \bmod N.$$

Dabei sind die Konstanten C_1 und C_2 so zu wählen, daß alle Werte von n erfaßt werden und Eindeutigkeit verhanden ist, d.h. jedes n durch eine bestimmte Kombination (n_1, n_2) erhalten wird. Derartige Kombinationen existieren jedoch immer [3.8]. Entsprechend wird der Laufindex k im Transformationsbereich durch

$$k = (C_3 k_1 + C_4 k_2) \bmod N$$

dargestellt. Eine Eindeutigkeit erhält man für

$$C_1 = a \cdot N_2, \quad C_2 = b \cdot N_1, \quad GGT(C_1, N_1) = GGT(C_2, N_2) = 1,$$

wobei a,b natürliche Zahlen sind. Entsprechendes gilt für C_3 und C_4 [3.8].

Falls die N_i nicht teilerfremd sind, z.B. $N=p^r$ (vgl. (3.11)), müssen für eine Eindeutigkeit die folgenden Bedingungen erfüllt sein [3.8]

$$C_1 = a \cdot N_2 \text{ und } C_2 \neq b \cdot N_1 \text{ und } GGT(a, N_1) = GGT(C_2, N_1) = 1$$

oder

$$C_1 \neq a \cdot N_2 \text{ und } C_2 = b \cdot N_1 \text{ und } GGT(C_1, N_1) = GGT(b, N_2) = 1.$$

Um eine Reduktion des Rechenaufwandes zu erhalten, sind nun die Konstanten so zu wählen, daß

$$(C_1 C_4) \bmod N = 0 \quad \text{oder} \quad (C_2 C_3) \bmod N = 0. \tag{3.12}$$

Wenn die N_i teilerfremd sind, können beide Bedingungen (3.12) erfüllt werden, wodurch zusätzlich Vereinfachungen in der Arithmetik möglich werden.

Die Abbildung der Indizes n und k auf die Subindizes n_i und k_j läßt die Philosophie der "schnellen" Algorithmen erkennen. Sie erlaubt die Aufspaltung einer großen Aufgabe in eine Reihe von kleineren. Das drückt sich so aus, daß in den Summenformen (1.2) die Summen über die n bzw. k durch mehrere unabhängige Summen über die n_i bzw. k_j ersetzt werden (vgl. Abschnitt 4.2.1). Eine andere Deutung dieser Abbildung ist die Umsetzung eines eindimensionalen Problems in ein mehrdimensionales mit jeweils unabhängigen Variablen.

Bei den Primfaktor-Algorithmen gemäß (3.10) bezeichnet man die Teilsummationen als Module. Im Fall der Radix-p-Transformationen nach (3.11) heißen die unabhängigen Teile Iterationen.

3.4.2 Darstellung durch Signalflußgraphen

Im Abschnitt 3.3 wurde am Beispiel der $(WHT)_H$ gezeigt, wie man durch geeignete Matrixfaktorisierung die Komplexität von $O(N^2)$ auf $O(N\log N)$ reduzieren kann. Die Faktorisierung erlaubt es, den Datenvektor zunächst mit der rechten Faktormatrix $[T]_1$, den Ergebnisvektor sodann mit der Faktormatrix $[T]_2$ zu multiplizieren, und so fort:

1. Verarbeitungsstufe: $[T]_1\ \underline{x}$

2. Verarbeitungsstufe: $[T]_2([T]_1\ \underline{x})$

.

.

.

log(N)-te Verarbeitungsstufe: $[T]_{\log N}([T]_{\log N-1} \cdots [T]_2[T]_1\ \underline{x})$

Ergebnis: $\underline{X} = [T]_{\log N}\ [T]_{\log N-1} \cdots [T]_2\ [T]_1\ \underline{x}$

Sofern N eine ganzzahlige Potenz einer Zahl $p \in \{2,3,\ldots\}$ ist, ergibt die Faktorisierung genau $\log_p N$ Faktormatrizen. Weiterhin beobachtet man, daß sowohl bei der Kronecker-Produktbildung als bei der Zerlegung in schwach besetzte Matrizen jedes neue Vektorelement eine gewichtete Summe aus dem entsprechenden Vektorelement der vorhergehenden Verarbeitungsstufe und (p-1) anderen Elementen des bisherigen Vektors ist (vgl. Abschnitt 3.3). Sofern eine Faktorisierung dieser Art möglich ist, kann man die Multiplikation des Datenvektors mit den Faktormatrizen nacheinander durchführen, derart daß nur genau p Zwischenspeicher erforderlich werden. Dies ist deshalb möglich, weil jeweils p Daten kombiniert und in ihre ursprünglichen Speicherplätze zurückgeschrieben werden können, denn die Ursprungsdaten werden im weiteren Verlauf nicht mehr benötigt. Die Matrixfaktorisierung ergibt in diesem Fall neben der beträchtlichen Komplexitätsminderung auch eine Begrenzung des Speicher- platzes auf N+p Wörter. Man spricht deshalb von einer Verarbeitung mit Ersetzung (der Speicherinhalte) oder einer "in-place"-Verarbeitung, Charak-

teristisch hierfür ist die Berechnung in Stufen oder Iterationen. Diese Verarbeitungsart kann man als die eigentliche Ursache für die Einsparung an Operationen ansehen: In jeder Iteration werden (gewichtete) Teilsummen gebildet, die in der darauf folgenden Iteration p-fach weiterverwendet werden.

Es ist üblich und zweckmäßig, die Verknüpfung der Eingangsdaten und der Zwischenergebnisse durch Signalflußgraphen darzustellen. Wegen der Aufteilung in Iterationen und wegen der "in-place"-Verarbeitung entstehen Signalflußgraphen (SFG) mit großer Regelmäßigkeit. Die Form der Radix-2-SFG (vgl. Bild 3.3) führte zu der Bezeichnung "Schmetterlings-Diagramm". Die Speicherplätze für Datenwörter werden im SFG meist vertikal angeordnet, die Iterationen horizontal. Auf einer horizontalen Linie liegende Knoten (vgl. z.B. Bild 3.3(b)) stellen also den gleichen Speicherplatz dar. Zu Beginn werden die Daten geladen (linke Seite des SFG) und nach Ablauf sämtlicher Iterationen enthält jeder Speicherplatz ein Element des transformierten Vektors (rechte Seite des SFG). Zwischen den einzelnen Iterationen sind in den Speicherplätzen Zwischenergebnisse enthalten, die manchmal von Interesse sein können.

Für die graphische Darstellung der SFG haben sich zwei verschiedene Darstellungsformen herausgebildet. Ihre Anwendung hängt von der Art des darzustellenden SFG ab. Die im Bild 3.3(a) gezeigte Form ist sehr allgemein, hat aber den Nachteil, daß umfangreiche SFG leicht unübersichtlich werden. Die Kreise stellen Multiplikationen mit den Konstanten a bis d dar. Eine Zusammenführung von 2 Kanten bedeutet die Addition der Signale. Für umfangreiche SFG empfiehlt sich die im Bild 3.3(b) gezeigte Darstellung. Allerdings ist ihre Anwendung auf SFG der FFT beschränkt.

In diesem Buch verwenden wir vorzugsweise eine Darstellung nach Bild 3.3(c). Dabei werden multiplikative Konstanten an der betreffenden Kante des SFG vermerkt. Gelegentlich wird der Faktor (-1), d.h. eine Subtraktion des Speicherinhalts durch eine gestrichelte Kante dargestellt (vgl. Bild 3.3(a)).

Die überwiegende Mehrzahl der im Buch beschriebenen Algorithmen verwendet die Radix-2-Struktur. Ein Radix-2-SFG für $N=2^r$ diskrete Daten besteht aus r Iterationen, in denen jeweils 2 in den Knoten (Speicherplätzen) enthaltene Werte linear kombiniert und (bei "in-place"-Verarbeitung) in diese Speicherplätze zurückgeschrieben werden. Dabei werden in den einzelnen Iteratio-

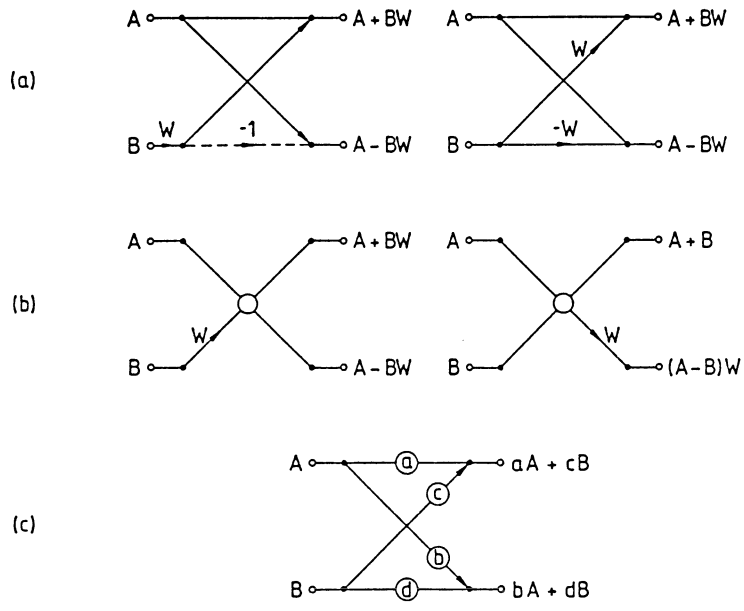


Bild 3.3. Darstellung von Radix-2-Signalflußgraphen in drei Formen

nen die Knoten gruppenweise verarbeitet: In einer Iteration werden jeweils zwei benachbarte Knoten kombiniert, in weiteren Iterationen werden Gruppen zu 4, 8, 16, ..., 2^r Knoten gebildet, wobei die Kombination zwischen Knotenpaaren mit den Distanzen 2, 4, 8, ..., 2^{r-1} erfolgt. Die Reihenfolge dieser Iterationen ist maßgebend für die Reihenfolge der Eingangs- und Ausgangswerte (vgl. den Abschnitt 3.4.3 und z.B. die Bilder 3.5 und 3.6).

SFG höherer Radizes sind analog zum Radix-2-SFG strukturiert. Bei einem SFG des Radix p werden jeweils Gruppen von p Speicherplätzen (Knoten) miteinander kombiniert. Von jedem Knoten gehen also genau p Kanten ab und es münden p Kanten in ihn ein. Bild 3.4(a) zeigt einen Radix-3-SFG, Bild 3.4(b) einen Radix-4-SFG. Aufgrund des SFG kann man die Komplexität einer Radix-p-Transformation gut übersehen. Wenn man $N=p^r$ Daten zu verarbeiten hat, benötigt man $r=\log_p N$ Iterationen. In jeder Iteration benötigt man maximal $p \cdot N$ Multiplikationen und $(p-1)N$ Additionen. Daraus folgt für die Komplexität:

$$K = O(N \log_p N).$$

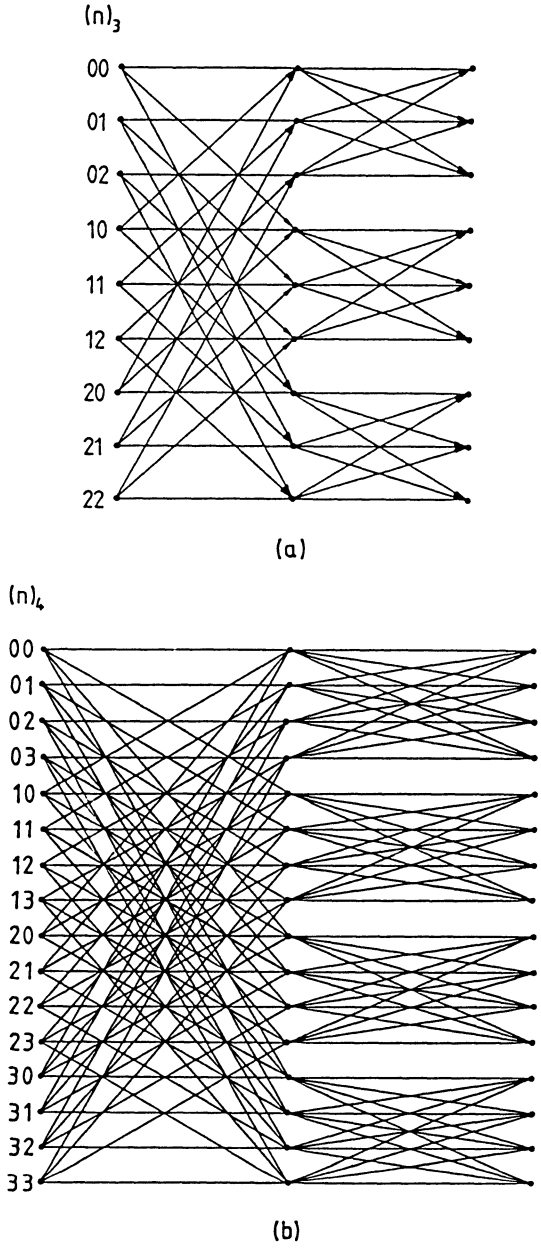


Bild 3.4. a) Radix-3-Signalflußgraph, b) Radix-4-Signalflußgraph

Wieviele Multiplikationen tatsächlich benötigt werden, ist eine Frage des jeweiligen Algorithmus', da die Multiplikatoren zum großen Teil trivial (z.B. 1, -1, j) sind. Durch geeignete Darstellung und geschickte Zusammenfassungen gelingt es, für bestimmte Transformationen die Anzahl der Multiplikationen zu verringern. Allerdings betragen die möglichen Einsparungen nur Bruchteile einer Größenordnung. Die Mindestanzahl von Multiplikationen für einen bestimmten Algorithmus läßt sich theoretisch ermitteln [3.9], [3.10].

Sofern die schmetterlingsartige Struktur der SFG in irgendeiner Weise durchbrochen wird, geht die "in-place"-Eigenschaft verloren. Dies tritt z.B. dann ein, wenn man verlangt, daß Eingangs- und Ausgangswerte eine bestimmte, dem Algorithmus nicht entsprechende Reihenfolge aufweisen müssen. Ein anderes Beispiel eines SFG ohne "in-place"-Eigenschaften ist ein solcher mit konstanter Geometrie in allen Iterationen (z.B. Bild 7.3). Derartige Verarbeitungen bieten gelegentlich Vorteile bei der Realisierung durch eine Spezial-Hardware.

3.4.3 Beziehungen zwischen Iterationenfolge, Objektdaten- und Koeffizientenreihenfolge

Wie im Abschnitt 3.3 gezeigt wurde, können geeignet faktorisierte Transformationsmatrizen in Signalflußgraphen umgesetzt werden. Für die am häufigsten verwendete Radix-2-Struktur hat man dann $r=\log_2 N$ Iterationen auszuführen. In diesem Abschnitt soll gezeigt werden, wie sich bei Umordnen der Reihenfolge von Iterationen die Struktur der Eingangs- und Ausgangswerte, d.h. der Objektdaten und der Koeffizienten ändert. Dabei wird angenommen, daß nur die Strukturen der SFG geändert werden, etwaige Multiplikatoren (z.B. Drehfaktoren bei der FFT) jedoch in der ursprünglichen Iteration verbleiben.

Zur Vereinfachung der Erläuterung diskutieren wir zunächst die Möglichkeiten der Anordnungen eines Signalflußgraphen für eine Radix-2-Struktur mit r Iterationen.

Wir können die Anzahl der Anordungsmöglichkeiten von r Iterationen durch die Permutation P_r darstellen:

$$P_r = r \cdot (r-1) \cdot (r-1) \cdots 3 \cdot 2 \cdot 1 = r!,$$

d.h. es gibt $P_r = r!$ Anordnungsmöglichkeiten für einen SFG mit r Iterationen. Nun betrachten wir ein Beispiel mit r=4:

Seien die vier Iterationen im SFG durch A, B, C und D dargestellt. Dann gibt es insgesamt $P_4 = 4! = 24$ Anordnungsmöglichkeiten der Iterationen, nämlich

ABCD,	BCDA,	CDAB,	DABC,
BADC,	ADCB,	DCBA,	CBAD,
CDBA,	DBAC,	DBCA,	BACD,
DCAB,	CABD,	CADB,	ABDC.

Es ist nun von Interesse, welche Auswirkung die Vertauschungen auf die Reihenfolge der Objektdaten (d.h. der Eingangswerte) und der Koeffizienten (Ausgangswerte) haben. Hierzu betrachten wir die Bitumkehroperation von n BRO=R(n), die im Abschnitt 2.4 bereits benutzt wurde. Wenn wir beispielsweise N=16 annehmen, dann ist r=4 und wir können k und n als 4-Bit-Binärzahlen darstellen. Das Bitumkehrverfahren ist durch Bild 2.5 illustriert.

Wir können in ähnlicher Weise wie im Bild 2.5 Bitaustauschoperationen (Bit Exchange Operations) BEO(1) bis BEO(P_r) definieren. In den folgenden Beispielen betrachten wir allerdings nur paarweise Vertauschungen von Iterationen. Nehmen wir beispielsweise an, der Index von $x(n_3,n_2,n_1,n_0)$ sei in der natürlichen Reihenfolge geordnet. Analog zu Bild 2.5 ist dann

$$BRO = R(n) = BEO(1) : \quad x(n_3,n_2,n_1,n_0) \longrightarrow x(n_0,n_1,n_2,n_3)$$

Entsprechend werden BEO(2) bis BEO(9) definiert. Die so definierten BEO-Möglichkeiten für N=4 sind in Tabelle 3.1 angegeben. Wenn man beispielsweise die Eingangswerte im SFG einer 16-Punkte-FFT gemäß Bild 3.5 nach BEO(1) umordnet, ergibt sich ein Signalflußgraph wie im Bild 3.6 dargestellt, d.h. mit der üblichen Bitumkehr.

Tabelle 3.1. Möglichkeiten der Indizierung, N=16: Die natürliche Reihenfolge und einige Versionen mit Bitaustausch sind durch BEO(0) bzw. durch BEO(m), (m = 1,2,....,9) bezeichnet.

BEO(0)	BEO(1)	BEO(2)	BEO(3)	BEO(4)
0000	0000	0000	0000	0000
0001	1000	0001	0010	1000
0010	0100	0010	0001	0010
0011	1100	0011	0011	1010
0100	0010	1000	0100	0100
0101	1010	1001	0110	1100
0110	0110	1010	0101	0110
0111	1110	1011	0111	1110
1000	0001	0100	1000	0001
1001	1001	0101	1010	1001
1011	0101	0110	1001	0011
1011	1101	0111	1011	1011
1100	0011	1100	1100	0101
1101	1011	1101	1110	1101
1110	0111	1110	1101	0111
1111	1111	1111	1111	1111

BEO(5)	BEO(6)	BEO(7)	BEO(8)	BEO(9)
0000	0000	0000	0000	0000
0001	0100	0001	0010	0100
0100	0010	1000	0001	1000
0101	0110	1001	0011	1100
0010	0001	0100	1000	0001
0011	0101	0101	1010	0101
0110	0011	1100	1001	1001
0111	0111	1101	1011	1101
1000	1000	0010	0100	0010
1001	1100	0011	0110	0110
1100	1010	1010	0101	1010
1101	1110	1011	0111	1110
1010	1001	0110	1100	0011
1011	1101	0111	1110	0111
1110	1011	1110	1101	1011
1111	1111	1111	1111	1111

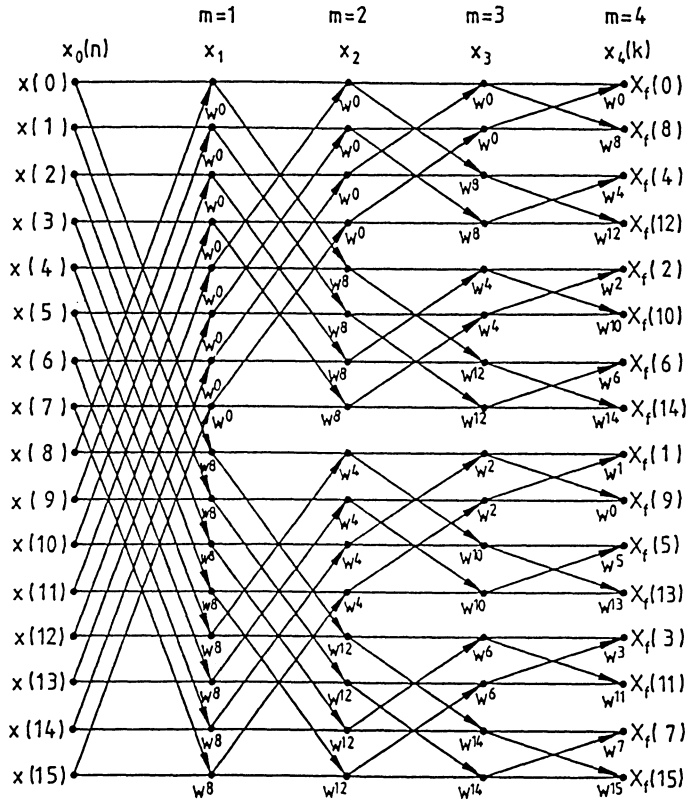


Bild 3.5. Signalflußgraph der FFT mit natürlicher Ordnung der Eingangswerte für N=16, (r=4)

$$x(n_3,n_2,n_1,n_0)$$

$$\downarrow$$

$$BEO(1)$$

$$\downarrow$$

$$x(n_0,n_1,n_2,n_3)$$

$$\downarrow$$

Bild 3.6

$$\downarrow$$

$$X(k_0,k_1,k_2,k_3)$$

Analog dazu können wir weitere Formen von Signalflußgraphen angeben, je nachdem wie man die Eingangswerte anordnet. Im Bild 3.7 sind wegen BEO(8)

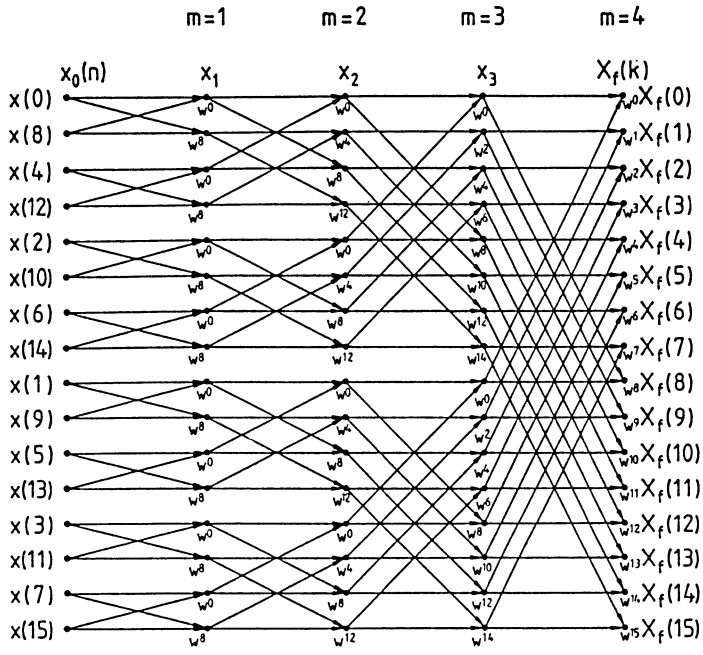


Bild 3.6. FFT-Signalflußgraph mit BEO(1)-Umordnung der Eingangswerte gegenüber Bild 3.5

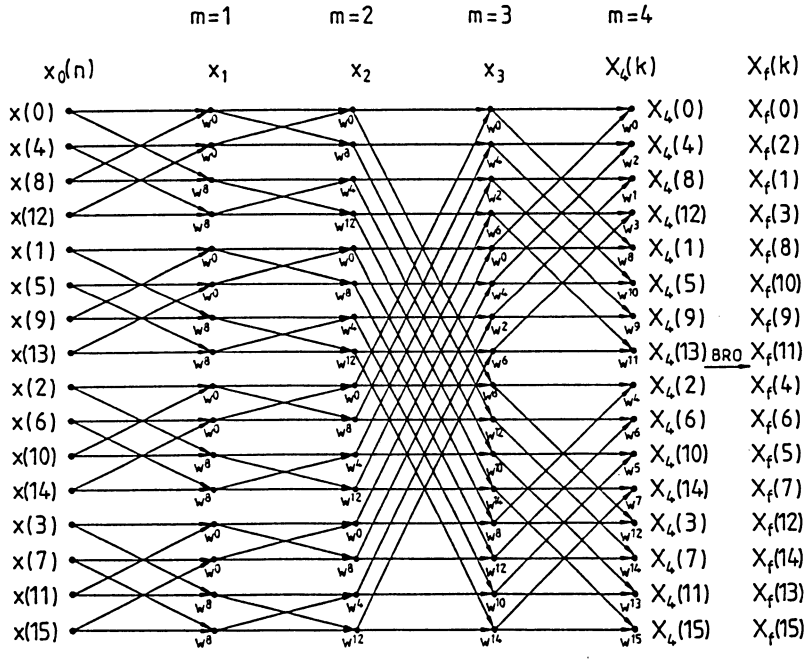


Bild 3.7. FFT-Signalflußgraph mit BEO(8)-Umordnung der Eingangswerte gegenüber Bild 3.5

(vgl. Tabelle 3.1) der Eingangswerte die Strukturen der 1. und der 2. Iteration sowie der 3. und der 4. Iteration miteinander vertauscht.

Wir untersuchen nun den Einfluß der Bitaustauschoperationen BEO auf die Folge der Iterationen und der Datensequenz. Dabei benutzen wir N=16 Datenwerte und entsprechend r=4 Iterationen als Beispiel. Die Unterschiede in der Struktur der einzelnen Iterationen eines Signalflußgraphen liegen im Abstand der dualen Knoten. Das sind solche Knoten (Speicherplätze), die in der betreffenden Iteration miteinander verknüpft werden. Die Abstände der dualen Knoten betragen für die a-te Iteration $N/2^a$ mit $a = 1,2,\ldots,r$. Betrachtet man die Ordnungszahl n der Knoten in binärer Darstellung

$$n = (n_{r-1}\ n_{r-2}\ \ldots\ n_1\ n_0)_2 = 0,1,\ldots,N-1,$$

so stellt man fest, daß jeweils solche Knoten verknüpft werden, deren Hamming-Distanz 1 beträgt und bei denen sich das unterschiedliche Bit in der b-ten Binärstelle befindet (vgl. Bild 3.5).

Für die 1.Iteration (a=1) gilt z.B.: $n_3\ n_2\ n_1\ n_0$

$x_1(8) \longrightarrow x_1(\ 1\ 0\ 0\ 0\ )$

$x_1(0) \longrightarrow x_1(\ 0\ 0\ 0\ 0\ )$

Differenz der Indizes: 8 $\longrightarrow$ 1 0 0 0

Binärstelle b: $\longrightarrow$ 1 2 3 4

Allgemein gilt für den SFG nach Bild 3.5, daß die binär geschriebene Differenz der Indizes eines dualen Knotenpaars in der a-ten Iteration nur in der Stelle b=a den Wert 1 hat. Alle anderen Stellen sind Null. Vertauscht man nun im der Binärindex der Eingangswerte die 1. Stelle mit der 4. Stelle und die 2. Stelle mit der 3. Stelle, dann erhält man die umgeordneten Eingangsdaten wie folgt:

$$x(\ n_3\ \ n_2\ \ n_1\ \ n_0\ )$$

$$\downarrow\ \ \downarrow\ \ \downarrow\ \ \downarrow$$

$$b{=}1\ \ b{=}2\ \ b{=}3\ \ b{=}4, \qquad a{=}b.$$

Dabei ist a die Ordnungszahl der Iteration im ursprünglichen Signalflußgraph (z.B. Bild 3.5). Wenn man die Eingänge z.B. nach BEO(9) (vgl. Tabelle 3.1) anordnet, dann erhält man für a=b:

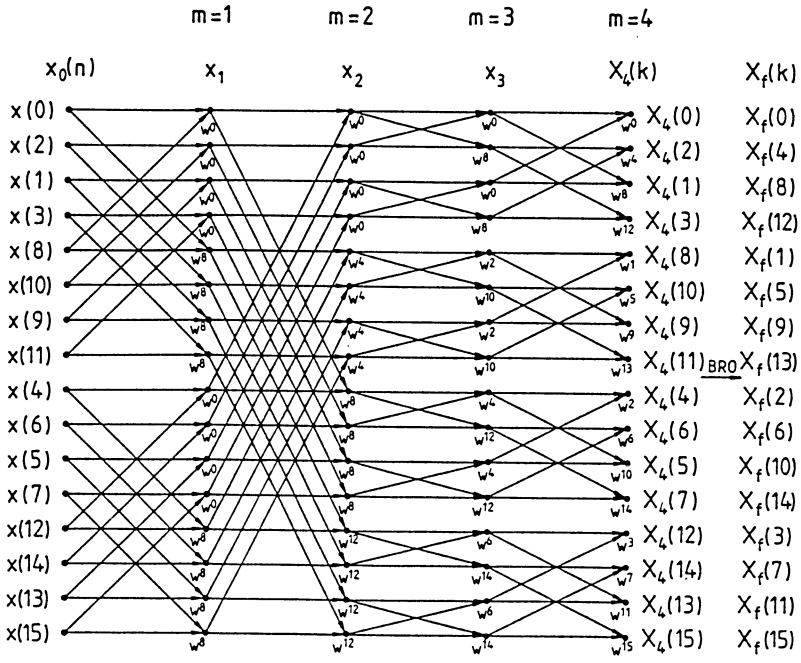


Bild 3.8. FFT-Signalflußgraph mit BEO(9)-Umordnung der Eingangswerte gegenüber Bild 3.5

$$x(\,n_1\;\;n_0\;\;n_3\;\;n_2\,)$$
$$\downarrow\quad\downarrow\quad\downarrow\quad\downarrow$$
$$b{=}3\quad b{=}4\quad b{=}1\quad b{=}2, \qquad a{=}b.$$

Der zugehörige Signalflußgraph ist im Bild 3.8 dargestellt. Die Ausführung der Transformation ergibt die Koeffizienten X_4, die sich durch Bitumkehr in die natürliche Ordnung überführen lassen:

$$X_4(k_3,k_2,k_1,k_0) \xrightarrow{\text{BRO}} X(k_0,k_1,k_2,k_3)$$

Schematisiert stellt sich der Zusammenhang für dieses Beispiel wie folgt dar:

$$x(n_3,n_2,n_1,n_0) \xrightarrow{\text{BEO(9)}} x(n_1,n_0,n_3,n_2)$$
$$\downarrow$$
$$\text{Signalflußgraph (Bild 3.8)}$$
$$\downarrow$$
$$X_4(k_3,k_2,k_1,k_0) \xrightarrow{\text{BRO}} X(k_0,k_1,k_2,k_3)$$

Entsprechendes gilt für die Signalflußgraphen der Bilder 3.6 und 3.7, wobei Bild 3.8 die bekannte einfache Bitumkehr darstellt. Um den Einfluß von Vertauschungen der Iterationen anschaulich darzustellen, wird der Zusammenhang wie folgt beschrieben:

$$x(n_{r-1},\ldots,n_i,\ldots,n_j,\ldots,n_1,n_0)$$

$$\downarrow$$

$$BEO(k) = \langle n_i : n_j \rangle$$

$$\downarrow$$

$$x(n_{r-1},\ldots,n_j,\ldots,n_i,\ldots,n_1,n_0)$$

$$\downarrow$$

mit entsprechend verändertem SFG

$$\downarrow$$

$$X(k_{r-1},\ldots,k_j,\ldots,k_i,\ldots,k_1,k_0)$$

$$\downarrow$$

$$BEO(1) = BRO \text{ (Bitumkehroperation)}$$

$$\downarrow$$

$$X_f(k_{r-1},\ldots,k_i,\ldots,k_j,\ldots,k_1,k_0)$$

Die o.g. Maßnahmen zur Veränderung der Datenreihenfolge haben Auswirkungen auf die Implementierung der Algorithmen. Insbesondere für die Verarbeitung durch Vektorrechner können hiermit günstige Arbeitsbedingungen geschaffen werden [3.11].

Wir betrachten nun noch die Auswirkungen von Vertauschungen der Iterationen einschließlich ihrer Multiplikatoren. Dabei beschränken wir uns der besseren Übersicht wegen auf 3 Iterationen eines Radix-2-SFG, d.h. N=8. Die Ergebnisse können leicht auf beliebig große Signalflußgraphen übertragen werden. Ausgangspunkt der Betrachtungen sei ein SFG gemäß Bild 3.9. Die Multiplikatoren a_i^j, b_i^j, c_i^j und d_i^j ($i = 1,2,\ldots,r = \log_2 N$, $j = 0,1,\ldots,N-1$) können von Knoten zu Knoten und von Iteration zu Iteration verschieden sein. Bei den hier betrachteten Orthogonaltransformationen herrscht allerdings eine gewisse Regelmäßigkeit vor. Zudem sind viele der Multiplikatoren gleich 1.

Wie wir in den Abschnitten 3.3 und 3.4.2 gezeigt haben, entspricht jede Iteration einer Multiplikation mit einer (schwach besetzten) Teilmatrix $[T_i]$.

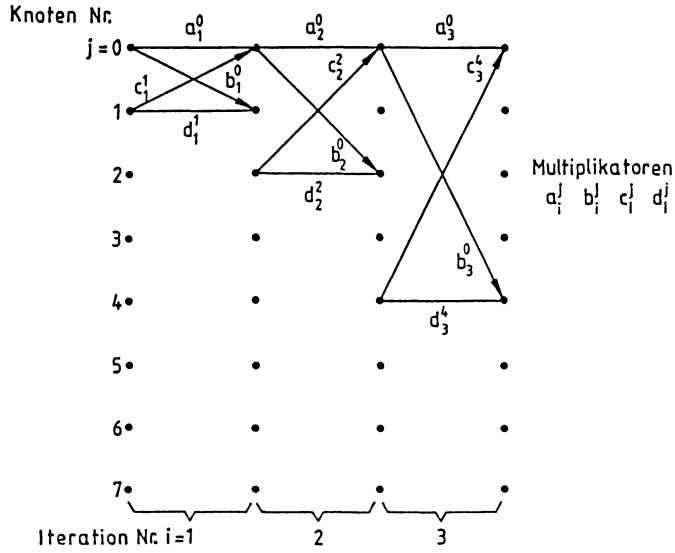


Bild 3.9. Signalflußgraph mit allgemeinen Multiplikatoren für N=8

Für den SFG nach Bild 3.9 erhalten wir die Transformationsmatrix

$$[T] = [T_3][T_2][T_1]$$

gemäß Bild 3.10. Eine Vertauschung der Iterationen einschließlich der zugehörigen Multiplikatoren entspricht einer Vertauschung der Reihenfolge der Faktoren $[T_i]$. Abgesehen von den Trivialfällen der Multiplikationen mit sich selbst oder mit der Einheitsmatrix ist die Matrizenmultiplikation genau dann kommutativ, wenn die Faktormatrizen symmetrisch sind, d.h. $b_i^j = c_i^k$. Das trifft bei der $(WHT)_H$ zu, nicht jedoch bei der DFT. Somit können die Iterationen der $(WHT)_H$ in beliebiger Reihenfolge ausgeführt werden. Diese nützliche Eigenschaft kann jedoch nicht auf andere Orthogonaltransformationen übertragen werden.

$$[T] = [T_3]\,[T_2]\,[T_1]$$

$$= \begin{bmatrix} a_3^0 & 0 & 0 & c_3^4 & 0 & 0 \\ 0 & a_3^1 & 0 & 0 & c_3^5 & 0 \\ 0 & 0 & a_3^2 & 0 & 0 & c_3^6 \\ 0 & 0 & 0 & a_3^3 & 0 & 0 \\ b_3^0 & 0 & 0 & d_3^4 & 0 & 0 \\ 0 & b_3^1 & 0 & 0 & d_3^5 & 0 \\ 0 & 0 & b_3^2 & 0 & 0 & d_3^6 \\ 0 & 0 & 0 & b_3^3 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_2^0 & c_2^2 & 0 & 0 & 0 & 0 \\ 0 & a_2^1 & c_2^3 & 0 & 0 & 0 \\ b_2^0 & d_2^2 & 0 & 0 & 0 & 0 \\ 0 & b_2^1 & d_2^3 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_2^4 & c_2^6 & 0 \\ 0 & 0 & 0 & 0 & a_2^5 & c_2^7 \\ 0 & 0 & 0 & b_2^4 & d_2^6 & 0 \\ 0 & 0 & 0 & 0 & b_2^5 & d_2^7 \end{bmatrix} \cdot \begin{bmatrix} a_1^0 & c_1^1 & 0 & 0 & 0 & 0 & 0 \\ b_1^0 & d_1^1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_1^2 & c_1^3 & 0 & 0 & 0 \\ 0 & 0 & a_1^2 & c_1^3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_1^4 & c_1^5 & 0 \\ 0 & 0 & 0 & 0 & b_1^4 & d_1^5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_1^6 c_1^7 \\ 0 & 0 & 0 & 0 & 0 & 0 & b_1^6 d_1^7 \end{bmatrix}$$

$$= \begin{bmatrix} a_{1\,2\,3}^{0\,0\,0} & c_{1\,2\,3}^{1\,0\,0} & a_{1\,2\,3}^{2\,2\,0} & c_{1\,2\,3}^{3\,2\,0} & a_{1\,2\,3}^{4\,4\,4} & c_{1\,2\,3}^{5\,5\,4} & a_{1\,2\,3}^{6\,6\,4} & c_{1\,2\,3}^{7\,6\,4} \\ b_{1\,2\,3}^{0\,1\,1} & d_{1\,2\,3}^{1\,1\,1} & b_{1\,2\,3}^{2\,3\,1} & d_{1\,2\,3}^{3\,3\,1} & b_{1\,2\,3}^{4\,5\,5} & d_{1\,2\,3}^{5\,5\,5} & b_{1\,2\,3}^{6\,7\,5} & d_{1\,2\,3}^{7\,7\,5} \\ a_{1\,2\,3}^{0\,0\,2} & c_{1\,2\,3}^{1\,0\,2} & a_{1\,2\,3}^{2\,2\,2} & a_{1\,2\,3}^{3\,2\,2} & a_{1\,2\,3}^{4\,4\,6} & c_{1\,2\,3}^{5\,4\,6} & a_{1\,2\,3}^{6\,6\,6} & c_{1\,2\,3}^{7\,6\,6} \\ b_{1\,2\,3}^{0\,1\,3} & d_{1\,2\,3}^{1\,1\,3} & b_{1\,2\,3}^{2\,3\,3} & d_{1\,2\,3}^{3\,3\,3} & b_{1\,2\,3}^{4\,5\,7} & d_{1\,2\,3}^{5\,5\,7} & b_{1\,2\,3}^{6\,7\,7} & d_{1\,2\,3}^{7\,7\,7} \\ a_{1\,2\,3}^{0\,0\,0} & c_{1\,2\,3}^{1\,0\,0} & a_{1\,2\,3}^{2\,2\,0} & c_{1\,2\,3}^{3\,2\,0} & a_{1\,2\,3}^{4\,4\,4} & c_{1\,2\,3}^{5\,4\,4} & a_{1\,2\,3}^{6\,6\,4} & c_{1\,2\,3}^{7\,6\,4} \\ b_{1\,2\,3}^{0\,1\,1} & d_{1\,2\,3}^{1\,1\,1} & b_{1\,2\,3}^{2\,3\,1} & d_{1\,2\,3}^{3\,3\,1} & b_{1\,2\,3}^{4\,5\,5} & d_{1\,2\,3}^{5\,5\,5} & b_{1\,2\,3}^{6\,7\,5} & d_{1\,2\,3}^{7\,7\,5} \\ a_{1\,2\,3}^{0\,0\,2} & c_{1\,2\,3}^{1\,0\,2} & a_{1\,2\,3}^{2\,2\,2} & c_{1\,2\,3}^{3\,2\,2} & a_{1\,2\,3}^{4\,4\,6} & c_{1\,2\,3}^{5\,4\,6} & a_{1\,2\,3}^{6\,6\,6} & c_{1\,2\,3}^{7\,6\,6} \\ b_{1\,2\,3}^{0\,1\,3} & d_{1\,2\,3}^{1\,1\,3} & b_{1\,2\,3}^{2\,3\,3} & d_{1\,2\,3}^{3\,3\,3} & b_{1\,2\,3}^{4\,5\,7} & d_{1\,2\,3}^{5\,5\,7} & b_{1\,2\,3}^{6\,7\,7} & d_{1\,2\,3}^{7\,7\,7} \end{bmatrix}$$

Bild 3.10. Transformationsmatrizen gemäß Bild 3.9

4 Algorithmen eindimensionaler Transformationen

4.1 Algorithmen für die diskrete Walsh-Transformation

4.1.1 Algorithmen für die Paley-, Hadamard- und Walsh-Ordnung

Die Walsh–Funktionen sind ein vollständiges System von orthogonalen Funktionen. Sie nehmen nur die beiden Funktionswerte (+1) und (−1) an. Deshalb benötigt die Berechnung der diskreten Walsh–Transformation (DWHT) nur Additionen und Subtraktionen. Die DWHT ist deswegen von sich aus schneller als alle anderen Orthogonaltransformationen.

Wie wir im Abschnitt 2.4 gezeigt haben, existiert keine einheitlich geordnete Form für die Walsh–Funktionen. Im Gegensatz zu den komplexen Exponentialfunktionen, die nach ihrer Frequenz geordnet werden, hat man mindestens drei in verschiedener Weise geordnete Walsh–Funktionensysteme und daher mindestens drei Formen von Transformationsalgorithmen.

Hinsichtlich der Computerorganisation sind die natürlich oder nach Hadamard geordneten Transformationskoeffizienten am einfachsten zu berechnen, da die Transformationsmatrix als Kronecker–Potenz der einfachen Hadamard–Matrix $[H_H(1)]$ abgeleitet werden kann. Eine gewisse Schwierigkeit bei der Walsh–Hadamard–Transformation $(WHT)_H$ besteht darin, eine nach Sequenzen geordnete Ausgabe zu erzeugen, die für viele Verwendungen wünschenswert ist. Dazu ist es notwendig, eine Bitumkehroperation bei Eingabe oder Ausgabe zusammen mit einer Gray–Code–Umordnung der Ausgabe auszuführen (vgl. Abschnitt 3.4).

Der Algorithmus für die $(WHT)_H$ wurde bereits im Abschnitt 3.3 durch Faktorisierung der Transformationsmatrix hergeleitet. Wir beschränken uns deshalb an dieser Stelle auf die Darstellung als Radix–2–Signalflußgraph. Da in den

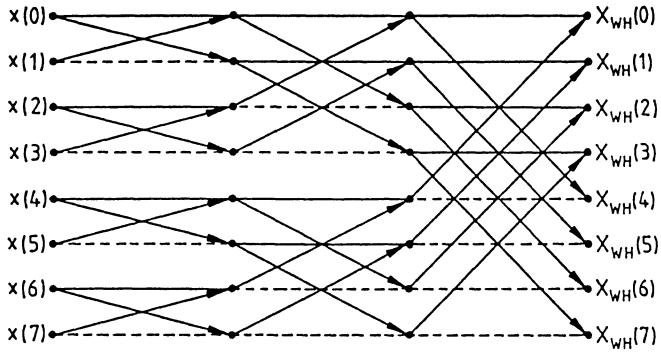


Bild 4.1. SFG einer natürlich geordneten Walsh-Transformation für r=3, nach [2.15]

Faktormatrizen $[T_i]$ in jeder Zeile (und jeder Spalte) nur p=2 von Null verschiedene Werte (+1 oder −1) auftreten, werden zur Bildung eines Zwischenwertes nur jeweils eine Addition oder eine Subtraktion benötigt. Weil diese Werte außerdem in jeweils anderen Zeilen (oder Spalten) stehen, können diese Operationen unabhängig von nachfolgenden Operationen stufen- oder iterationsweise durchgeführt werden. Mit den im Abschnitt 3.4.2 angegebenen Regeln erhält man somit einen sehr einfachen Signalflußgraph. Ein SFG für r=3 ist im Bild 4.1 dargestellt.

Da die natürliche oder Hadamard-Ordnung wegen der Darstellbarkeit der zugehörigen Matrix als Kronecker-Potenz von $[H_H(1)]$ am häufigsten benutzt wird, beziehen wir die anderen Anordnungen auf die natürliche Ordnung. In Tabelle 4.1 ist nochmal die Reihenfolge der Basisfunktionen (und damit der Koeffizienten) bezogen auf die natürliche Ordnung angegeben. Diese Tabelle illustriert die Umordnung der Koeffizientenindizes für die verschiedenen Ordnungen. Man ersieht aus ihr, daß sich die natürliche (Hadamard-) Ordnung von der dyadischen (Paley-) Ordnung durch Bitumkehr der Koeffizientenindizes ergibt. Gemäß den Ausführungen im Abschnitt 3.4 wird dies durch Umkehrung der Iterationenfolge erreicht (vgl. Bild 4.2). Entprechend erhält man den SFG der sequenzgeordneten Variante durch Bitumordnung und inverse Konvertierung des Gray-Codes. Der zugehörigen SFG für r=3 ist im Bild 4.3 angegeben.

Da die Knoten der Graphen mit Speicherplätzen im Rechner korrespondieren, ist aus den Abbildungen zu ersehen, daß für die Berechnung kein zusätzlicher

Tabelle 4.1. Ordnung der Walsh-Koeffizienten für r=3

Koeffizienten-Nummer	Hadamard-Ordnung	Walsh-Ordnung	Paley-Ordnung
0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 1 1	1 1 1 7	1 0 0 4
2	0 1 0 2	0 1 1 3	0 1 0 2
3	0 1 1 3	1 0 0 4	1 1 0 6
4	1 0 0 4	0 0 1 1	0 0 1 1
5	1 0 1 5	1 1 0 6	1 0 1 5
6	1 1 0 6	0 1 0 2	0 1 1 3
7	1 1 1 7	1 0 1 5	1 1 1 7

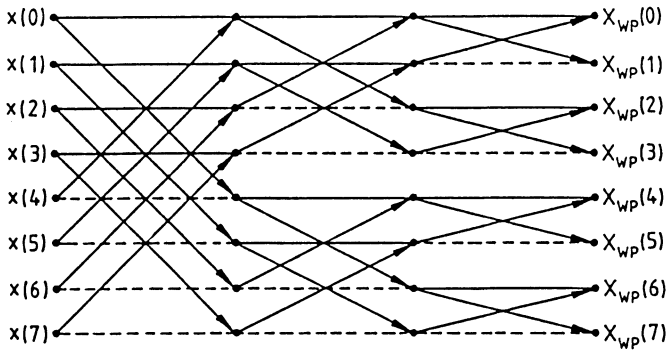


Bild 4.2. SFG einer dyadisch geordneten Walsh-Transformation für r=3, nach [2.15]

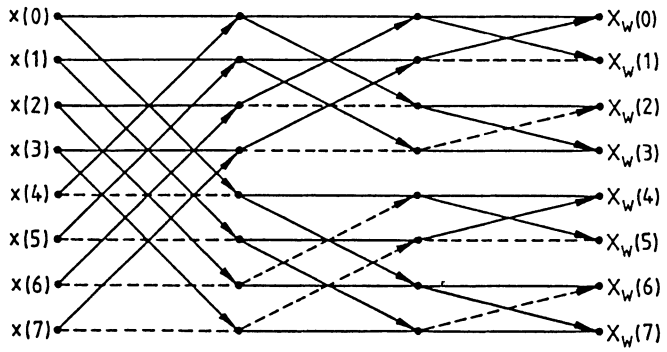


Bild 4.3. SFG einer sequenzgeordneten Walsh-Transformation für r=3, nach [2.15]

Speicherplatz benötigt wird ("in-place"). Es sind auch SFG ohne "in-place"-Eigenschaft bekannt, auf deren Darstellung wir jedoch verzichten.

4.1.2 Berechnung der WT aus Koeffizienten von Teilfolgen

Gelegentlich kann es zweckmäßig sein, zunächst die Koeffizienten von Teilen der Datenfolge zu berechnen und aus diesen schrittweise die Koeffizienten höherer Ordnung. Wir zeigen deshalb, daß eine Berechnung der WHT-Koeffizienten aus zwei Teilfolgen der Länge N/2 möglich ist. Der zu transformierende Datenvektor mit N Elementen ($N=2^r$) sei mit $\underline{x}$ bezeichnet, der Vektor seiner Walsh-Hadamard-Koeffizienten mit $\underline{X}_{WH}$. Dann haben wir

$$\underline{X}_{WH} = [H_H(L)]\,\underline{x}, \qquad L = 1,2,\ldots,r,$$

wobei $[H_H(L)]$ die Transformationsmatrix der Ordnung 2^L ist. Man kann den Vektor $\underline{x}$ der Ordnung N in zwei Vektoren der Ordnung N/2 zerlegen

$$\underline{x} = \begin{bmatrix} \underline{x}_1(L-1) \\ - - - - - \\ \underline{x}_2(L-1) \end{bmatrix}.$$

Dabei ist $\underline{x}_1(L-1)$ ein Vektor der Länge 2^{L-1}, der aus den N/2 oberen Elementen des Vektors $\underline{x}$ besteht, und $\underline{x}_2(L-1)$ ein solcher aus den unteren Elementen desselben Vektors.

Außerdem kann $\underline{X}_{WH}$ in der folgenden Weise zerlegt werden:

$$[X_{WH}(L)] = \begin{bmatrix} [H_H(L-1)] & [H_H(L-1)] \\ - - - - - - - - - - - - - - - \\ [H_H(L-1)] & -[H_H(L-1)] \end{bmatrix} \begin{bmatrix} \underline{x}_1(L-1) \\ - - - - - \\ \underline{x}_2(L-1) \end{bmatrix}. \tag{4.1}$$

Diese Matrixmultiplikation kann nun ausgeführt werden:

$$[X_{WH}(L)] = \begin{bmatrix} [H_H(L-1)]\underline{x}_1(L-1) + [H_H(L-1)]\underline{x}_2(L-1) \\ - \\ [H_H(L-1)]\underline{x}_1(L-1) - [H_H(L-1)]\underline{x}_2(L-1) \end{bmatrix} =$$

$$= \begin{bmatrix} \underline{X}_{WH1}(L-1) + \underline{X}_{WH2}(L-1) \\ - - - - - - - - - - - - - - - \\ \underline{X}_{WH1}(L-1) - \underline{X}_{WH2}(L-1) \end{bmatrix}. \qquad (4.2)$$

Dabei bezeichnen $\underline{X}_{WH1}(L-1)$ und $\underline{X}_{WH2}(L-1)$ die Walsh-Hadamard-Transformierten der oberen und der unteren Hälfte von $\underline{x}$. Diese Beziehungen kann man mit Vorteil auch zur Gewinnung der DFT-Koeffizienten und der DCT-Koeffizienten verwenden (vgl. die Abschnitte 4.2.6 und 4.4.6).

Durch wiederholte Anwendung von (4.2) kann man erreichen, daß zunächst die WH-Koeffizienten benachbarter Datenelemente gebildet werden, aus diesen dann die Koeffizienten benachbarter Datenpaare und so fort. Die stufenweise Bildung höher Koeffizienten aus untergeordneten bezeichnen wir als hierarchisch (vgl. die Abschnitte 3.1.2 sowie 5.4).

Die $(WHT)_H$ ist also eine orthogonale Transformation, die sich hierarchisch durchführen läßt. Die vorstehende Herleitung läßt aus (4.1) erkennen, daß hierfür die Transformationsmatrix durch eine Kronecker-*Potenz* (nicht Kronecker-*Produkt*) darstellbar sein muß. Außer der $(WHT)_H$ ist keine orthogonale Transformation bekannt, die eine Kronecker-Potenz-Darstellung besitzt. Also kann eine hierarchische Erzeugung von Transformationskoeffizienten nur über die WHT erfolgen.

4.2 Algorithmen für die diskrete Fourier-Transformation

Man kennt zahlreiche Algorithmen zur Berechnung der DFT. Das erste und wohl auch das bekannteste Verfahren zur schnellen Berechnung der DFT ist das von Cooley und Tukey [4.1], [4.2] veröffentlichte sogenannte FFT-Verfahren sowie eine andere von Sande entwickelte selbständige Form, der Sande-Tukey-FFT-Algorithmus. Geschichtlich ist zu bemerken, daß bereits 1958 von Good ein Algorithmus zur schnellen Berechnung der FFT angegeben wurde [4.3]. Da Good diesen Algorithmus für Zwecke der Statistik benötigte und auch in einer einschlägigen Zeitschrift veröffentlichte, wurde sein Algorithmus für die Signalverarbeitung zunächst nicht populär. Andererseits gab es auch schon vor Good diverse Ansätze zu solchen Algorithmen. So kann man schon bei Gauß eine Rechenvorschrift ähnlicher Art finden [4.4]. In Deutschland gab es in den

ersten Jahrzehnten dieses Jahrhunderts weitere Ansätze bezüglich einer schnellen Berechnung der Fourier-Transformation [4.5], [4.6].

Nachdem Cooley und Tukey die FFT in der Signalverarbeitung eingeführt hatten, wurden zahlreiche andere FFT-Algorithmen entwickelt, meist mit dem Ziel, die Anzahl der Multiplikationen zu verringern. So entstanden z.B. der Radix-2^r-Algorithmus, der Split-Radix-FFT-Algorithmus und der Primfaktor-Algorithmus (PFA). Unter diesen werden derzeit noch die Radix-2- und Radix-4-Algorithmen wegen ihrer einfachen SFG-Strukturen am häufigsten in der Praxis benutzt. Die zugehörigen Signalflußgraphen haben eine einfache Geometrie (Schmetterlings-Typ) und bieten die Möglichkeit zur Ausführung "mit Ersetzung" ("in-place"), sind aber aufwendiger bezüglich der Anzahl der Multiplikationen als einige neuere Algorithmen. Diese benötigen zwar weniger Multiplikationen als der ursprüngliche Cooley-Tukey-Algorithmus, erfordern dafür aber mehr Additionen. Andererseits hat man Algorithmen entwickelt, die zwar gleich viele Multiplikationen benötigen, jedoch eine reduzierte Anzahl von Additionen. Der Split-Radix-Algorithmus scheint einige der Vorteile dieser Methoden zu vereinigen. Die Vor- und Nachteile relativieren sich u.U., wenn man die Hardware in Betracht zieht, auf der die Algorithmen laufen sollen. Man vergleiche hierzu auch die Ausführungen im Kapitel 7.

Eine gänzlich andere Philosophie liegt dem Winograd-Fourier-Transformationsalgorithmus (WFTA) zugrunde. Bei diesem Algorithmus wird die DFT über den "Umweg" der zyklischen Faltung berechnet. Durch geeignete Maßnahmen gelingt es, mit beträchtlich weniger Multiplikationen auszukommen als für die "klassische" FFT benötigt werden.

Im allgemeinen wünscht man die Ermittlung sämtlicher Fourier-Koeffizienten. Die in diesem Buch beschriebenen FFT-Algorithmen sind geeignet, alle N Koeffizienten effizient zu berechnen. Falls nur ein (oder wenige) Koeffizient(en) benötigt wird (werden), wird die direkte Berechnung zweckmäßiger sein. Für derartige Sonderfälle existieren überdies spezielle Algorithmen wie z.B. der Görtzel-Algorithmus [3.9].

4.2.1 Cooley-Tukey- und Sande-Tukey-Algorithmus

Die FFT-Algorithmen von Cooley, Tukey und Sande sind nicht nur die ersten in der Geschichte der modernen Signalverarbeitung, sondern auch die am besten durchschaubaren. Sie zeigen die grundlegenden Prinzipien in einfacher Weise. Um dieses noch zu unterstreichen, führen wir sie zunächst anhand einfacher Spezialfälle, nämlich für Datensequenzen der Längen 4 und 8 ein. Eine allgemeinere Darstellung wird daran anschließend bei der Diskussion der Begriffe "Zeitdezimierung" und "Frequenzdezimierung" gegeben.

Wir gehen aus von der Summenform der diskreten Fourier-Transformation [2.1]

$$X_f(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad k = 0,1,\ldots,N-1. \tag{4.3}$$

Dabei benutzen wir die übliche Schreibweise $W_N = \exp(-j2\pi/N)$.

Für N=4 erhält man beispielsweise aus (4.3) die Gleichungen

$$\begin{aligned}
X_f(0) &= x(0)W_4^0 + x(1)W_4^0 + x(2)W_4^0 + x(3)W_4^0 \\
X_f(1) &= x(0)W_4^0 + x(1)W_4^1 + x(2)W_4^2 + x(3)W_4^3 \\
X_f(2) &= x(0)W_4^0 + x(1)W_4^2 + x(2)W_4^4 + x(3)W_4^6 \\
X_f(3) &= x(0)W_4^0 + x(1)W_4^3 + x(2)W_4^6 + x(3)W_4^9
\end{aligned}$$

oder kompakter in Matrixform

$$\underline{X}_f = [W_4^E]\,\underline{x} = \begin{bmatrix} X_f(0) \\ X_f(1) \\ \\ X_f(2) \\ X_f(3) \end{bmatrix} = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ & & & \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ \\ x(2) \\ x(3) \end{bmatrix}.$$

Für die Entwicklung des Algorithmus' stellen wir die natürlichen Zahlen n und k als Binärzahlen dar. Für N=4 ist r=2. k und n sind also 2-Bit-Binärzahlen

$$n = (n_1,n_0)_{bin} = \{00,01,10,11\}, \qquad k = (k_1,k_0)_{bin} = \{00,01,10,11\},$$

$$\text{d.h. } n = 2^1 n_1 + 2^0 n_0 \quad \text{und} \quad k = 2^1 k_1 + 2^0 k_0. \tag{4.4}$$

Mit (4.4) und (4.3) erhält man [1)]

$$X_f(k_1,k_0) = \sum_{n_0=0}^{1} \sum_{n_1=0}^{1} x_0(n_1,n_0)\, W_4^{(2n1+n0)(2k1+k0)}. \tag{4.5}$$

Darin kann eine Summation ausgeklammert werden, so daß (4.5) wie folgt dargestellt werden kann

$$X_f(k_1,k_0) = \sum_{n_0=0}^{1} \Big(\sum_{n_1=0}^{1} x_0(n_1,n_0)\, W_4^{(2k0n1)} \Big)\, W_4^{(2k1+k0)n0}. \tag{4.6}$$

Aus dieser Gleichung läßt sich der Cooley-Tukey-FFT-Algorithmus für N=4 herleiten. Zunächst betrachten wir die Summe in der Klammer

$$x_1(k_0,n_0) = \sum_{n_1=0}^{1} x_0(n_1,n_0)\, W_4^{(2k0n1)}. \tag{4.7}$$

Durch Einsetzen aller möglichen Zahlenwerte für k_0 und n_0 erhält man aus (4.7) die folgenden Gleichungen:

$$\begin{aligned}
x_1(0,0) &= x_0(0,0) + x_0(1,0)W_4^0,\\
x_1(0,1) &= x_0(0,1) + x_0(1,1)W_4^0,\\
x_1(1,0) &= x_0(0,0) + x_0(1,0)W_4^2,\\
x_1(1,1) &= x_0(0,1) + x_0(1,1)W_4^2.
\end{aligned} \tag{4.8}$$

Die Überführung in Matrixdarstellung ergibt

$$\begin{bmatrix} x_1(0,0) \\ x_1(0,1) \\ \\ x_1(1,0) \\ x_1(1,1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & W_4^0 & 0 \\ 0 & 1 & 0 & W_4^0 \\ & & & \\ 1 & 0 & W_4^2 & 0 \\ 0 & 1 & 0 & W_4^2 \end{bmatrix} \begin{bmatrix} x_0(0,0) \\ x_0(0,1) \\ \\ x_0(1,0) \\ x_0(1,1) \end{bmatrix}. \tag{4.9}$$

Entsprechend erhält man für die äußere Summe von (4.5)

$$x_2(k_0,k_1) = \sum_{n_0=0}^{1} x_1(k_0,n_0) W_4^{(2k1+k0)n0}. \tag{4.10}$$

---

1) Aus drucktechnischen Gründen werden Indizes der Exponenten auf der (hochgestellten) Zeile geschrieben.

oder in Matrixform

$$\begin{bmatrix} x_2(0,0) \\ x_2(0,1) \\ \\ x_2(1,0) \\ x_2(1,1) \end{bmatrix} = \begin{bmatrix} 1 & W_4^0 & 0 & 0 \\ 1 & W_4^2 & 0 & 0 \\ & & & \\ 0 & 0 & 1 & W_4^1 \\ 0 & 0 & 1 & W_4^3 \end{bmatrix} \begin{bmatrix} x_1(0,0) \\ x_1(1,0) \\ \\ x_1(1,0) \\ x_1(1,1) \end{bmatrix}.$$

Durch Einsetzung von (4.9) ergibt sich

$$\begin{bmatrix} x_2(0,0) \\ x_2(0,1) \\ \\ x_2(1,0) \\ x_2(1,1) \end{bmatrix} = [T_1][T_2] \begin{bmatrix} x_0(0,0) \\ x_0(0,1) \\ \\ x_0(1,0) \\ x_0(1,1) \end{bmatrix}. \qquad (4.11)$$

Dabei sind die Matrizen $[T_1]$ und $[T_2]$

$$[T_1] = \begin{bmatrix} 1 & W_4^0 & 0 & 0 \\ 1 & W_4^2 & 0 & 0 \\ & & & \\ 0 & 0 & 1 & W_4^1 \\ 0 & 0 & 1 & W_4^3 \end{bmatrix}, \qquad [T_2] = \begin{bmatrix} 1 & 0 & W_4^0 & 0 \\ 0 & 1 & 0 & W_4^0 \\ & & & \\ 1 & 0 & W_4^2 & 0 \\ 0 & 1 & 0 & W_4^2 \end{bmatrix}.$$

Aus (4.5), (4.6), (4.7) und (4.10) folgt schließlich

$$\underline{X}_f(k_1,k_0) = \underline{x}_2(k_0,k_1). \qquad (4.12)$$

Wie man durch Vergleich von $\underline{x}_2(k_0,k_1)$ mit $\underline{X}_f(k_1,k_0)$ erkennt, treten die Koeffizienten in der Bitumkehrreihenfolge auf.

Die Gleichungen (4.6) und (4.10) sind in sofern iterativ, als die zweite Gleichung ausschließlich die Ergebnisse aus der ersten Gleichung verwendet. Wir stellen nun (4.6) durch den im Bild 4.4 abgebildeten Signalflußgraph dar. Dazu wird der Vektor der Objektdaten $x_0(n_1,n_0)$ auf eine senkrechte Knotenspalte auf der linken Seite des Graphen abgebildet. Eine zweite Spalte beschreibt den nach (4.7) berechneten Vektor $\underline{x}_1(k_0,n_0)$ und schließlich er-

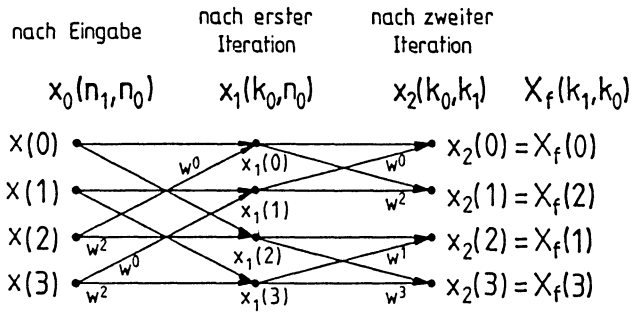


Bild 4.4. FFT-Signalflußgraph für N=4, nach [2.1]

hält man den Vektor $\underline{x}_2(k_0,k_1) = \underline{X}_f(k_1,k_0)$ gemäß (4.12) an der rechten Seite des Graphen.

Der Signalflußgraph beschreibt die Verarbeitung mit der faktorisierten Transformationsmatrix (4.11). Jede Iteration des Graphen entspricht einer Teilmatrix $[T_i]$ mit $i = 1,2,\ldots,r$. Für $N=2^r$ hat man r Spalten aus jeweils N Elementen.

Zur weiteren Erklärung der hierbei verwendeten Ausdrucksweise betrachten wir (4.3) für den Fall $N = 2^3 = 8$. Dann hat man folgende Binärdarstellungen für k und n:

$$k = 4\cdot k_2 + 2\cdot k_1 + k_0, \quad n = 4\cdot n_2 + 2\cdot n_1 + n_0, \quad k_i, n_i \in \{0,1\}, \quad i = 0,1,2.$$

Damit folgt aus (4.3)

$$X_f(k_2,k_1,k_0) = \sum_{n_2=0}^{1} \sum_{n_1=0}^{1} \sum_{n_0=0}^{1} x_0(n_2,n_1,n_0)\, W_8^{(4k_2+2k_1k_0)(4n_2+2n_1+n_0)}. \qquad (4.13)$$

Nun kann man (4.13) wie folgt umschreiben:

$$x_1(k_0,n_1,n_0) = \sum_{n_2=0}^{1} x_0(n_2,n_1,n_0) W_8^{4k_0n_2},$$

$$x_2(k_0,k_1,n_0) = \sum_{n_1=0}^{1} x_1(k_0,n_1,n_0) W_8^{(2k_1+k_0)2n_1},$$

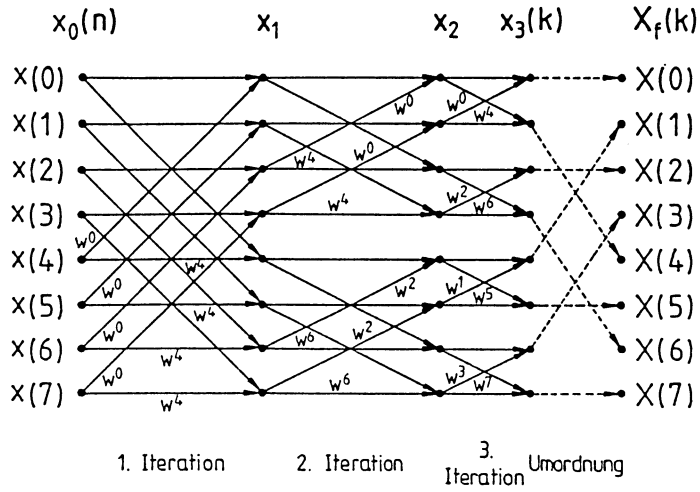


Bild 4.5. Cooley-Tukey-FFT-Signalflußgraph für N=8, nach [2.1]

$$X_3(k_2,k_1,k_0) = \sum_{n_0=0}^{1} x_2(k_0,k_1,n_0) W_8^{(4k_2+2k_1+k_0)n_0} \quad .$$

$$X_f(k_2,k_1,k_0) = X_3(k_0,k_1,k_2).$$

Damit erhält man die gewünschte Aufteilung in drei iterative Schritte (vgl. den Signalflußgraph Bild 4.5). Entsprechend ergibt sich z.B. der Signalflußgraph für N=16 (vgl. Bild 3.5).

Neben dem hier dargestellten Algorithmus, der ursprünglich von Cooley und Tukey entwickelt wurde und nach ihnen benannt ist, gibt es eine Reihe von Variationen dieses Algorithmus'. Diese können sich in speziellen Anwendungsfällen als geeigneter erweisen.

Wie aus vorstehendem Beispiel ersichtlich, ist eine Bitumkehrung von k zur Umordnung der Koeffizienten nötig. Man kann nun den Signalflußgraph aus Bild 4.5 derart modifizieren, daß die Eingangswerte in Bitumkehr-Reihenfolge und die Ausgangsgrößen in der natürlichen Reihenfolge auftreten. Wir verweisen hierzu auf Abschnitt 3.4.3, in dem der Zusammenhang zwischen der Reihenfolge der Eingangsdaten und der transformierten Daten durch "bit reversal", d.h.

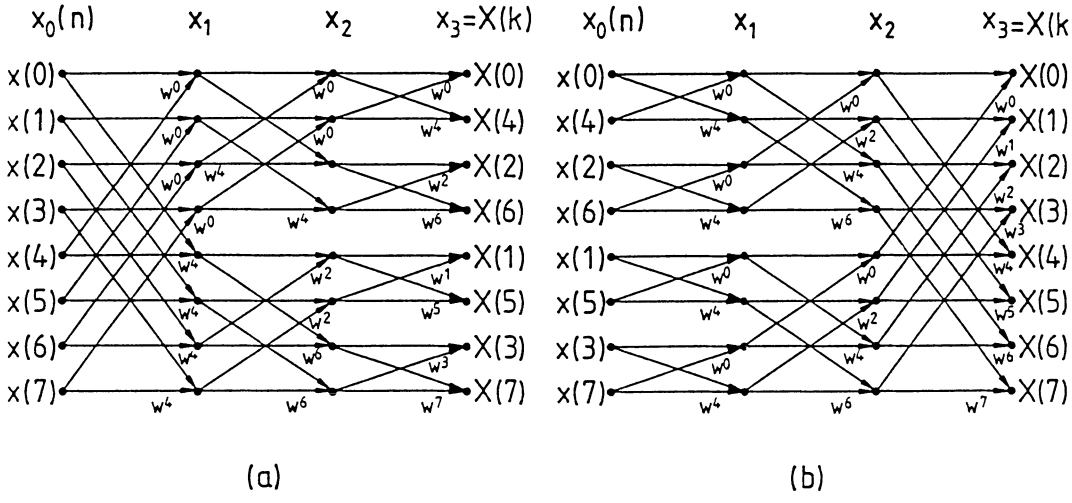


Bild 4.6. Cooley-Tukey-Signalflußgraphen: a) Eingangswerte in natürlicher Reihenfolge, b) Eingangswerte in Bitumkehrreihenfolge, nach [2.1]

durch Vertauschen der Iterationsspalten im Signalflußgraph ganz allgemein diskutiert wurde. Bild 4.6 zeigt die beiden Formen des Cooley-Tukey-Algorithmus': Natürliche Ordnung der Eingänge (a) bzw. der Ausgänge (b).

Der Cooley-Tukey-Algorithmus wird in der Literatur oft als FFT-Algorithmus mit Zeitdezimierung (decimation in time, DIT) bezeichnet. Zur Erläuterung dieser Bezeichnung betrachten wir die folgende Aufspaltung der FFT in Daten mit geraden und ungeraden Indexwerten:

$$X_f(k) = \sum_{n \text{ gerade}} x(n) W_N^{kn} + \sum_{n \text{ ungerade}} x(n) W_N^{kn}.$$

Seien durch $f(n) = x(2p)$ die gerade indizierten Werte von $x(n)$ sowie durch $g(n) = x(2p+1)$ die Werte mit ungeraden Indizes dargestellt, d.h. $n=2p$ für die erste Summe und $n = 2p+1$ für die zweite. Dann haben wir

$$X_f(k) = \sum_{p=0}^{(N/2)-1} x(2p) W_N^{2pk} + \sum_{p=0}^{(N/2)-1} x(2p+1) W_N^{(2p+1)k}.$$

Weil $W_N^2 = \exp(-j4\pi/N) = \exp(-j2\pi/(N/2)) = W_{N/2}$, erhält man

$$X_f(k) = \sum_{p=0}^{(N/2)-1} x(2p)W_{N/2}^{pk} + W_N^k \sum_{p=0}^{(N/2)-1} x(2p+1)W_{N/2}^{pk}$$

oder

$$X_f(k) = \begin{cases} F(k) + W_N^k G(k) & \text{für } 0 \leqq k \leqq (N/2)-1 \\ F(k-(N/2)) + W_N^k G(k-(N/2)) & \text{für } (N/2) \leqq k \leqq N-1. \end{cases}$$

Dabei sind F(k) und G(k) die DFT der Datensequenzen f(n) bzw. g(n).

Wegen $W_N^{k+(N/2)} = - W_N^k$ kann man schreiben

$$\left.\begin{aligned} F(k) + W_N^k G(k) &= X_f(k) \\ F(k) - W_N^k G(k) &= X_f(k+(N/2)). \end{aligned}\right\} \quad k = 0,1,\ldots,(N/2)-1$$

Dieses Verfahren wird rekursiv weiter durchgeführt und ergibt schließlich den Cooley-Tukey-Algorithmus mit

$$F(k) = \sum_{p=0}^{(N/4)-1} f(2p)W_{N/2}^{2pk} + \sum_{p=0}^{(N/4)-1} f(2p+1)W_{N/2}^{(2p+1)k}$$

$$= \sum_{p=0}^{(N/4)-1} f(2p)W_{N/4}^{pk}W_{N/2}^{k} + \sum_{p=0}^{(N/4)-1} f(2p+1)W_{N/4}^{pk}$$

und

$$G(k) = \sum_{p=0}^{(N/4)-1} g(2p)W_{N/4}^{pk}W_{N/2}^{k} + \sum_{p=0}^{(N/4)-1} g(2p+1)W_{N/4}^{pk} \quad .$$

Die Bezeichnung "mit Zeitdezimierung" müßte eigentlich lauten "mit Reduktion der Abtastraten im Zeitbereich". Der Begriff stammt also von der Anwendung der DFT auf Zeitfunktionen.

Der Sande-Tukey-Algorithmus ist strukturell ähnlich aufgebaut. Im Gegensatz zum Cooley-Tukey-Algorithmus wird bei diesem der Frequenzindex k in gerade und ungerade Werte aufgeteilt. Deshalb wird der Sande-Tukey-Algorithmus häufig auch als FFT-Algorithmus mit Frequenzdezimierung (decimation in frequency, DIF) bezeichnet. Der grundlegende Unterschied ergibt sich im Verlauf der Herleitung:

$$X_f(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} = \sum_{n=0}^{(N/2)-1} x(n)W_N^{nk} + \sum_{n=(N/2)}^{N-1} x(n+(N/2))W_N^{(n+(N/2))k}$$

$$= \sum_{n=0}^{(N/2)-1} x(n)W_N^{nk} + \sum_{n=0}^{(N/2)-1} x(n+(N/2))W_N^{nk}$$

$$= \sum_{n=0}^{(N/2)-1} \{x(n) + (-1)^k x(n+(N/2))\}\ W_N^{nk}.$$

Nun werde k in gerade und ungerade Werte aufgeteilt. Dann erhalten wir

$$X_f(2p) = \sum_{n=0}^{(N/2)-1} \{x(n) + x(n+(N/2))\}\ W_N^{2np}$$

und

$$X_f(2p+1) = \sum_{n=0}^{(N/2)-1} \{x(n) - x(n+(N/2))\}\ W_N^{2np}\ W_N^{n}$$

mit

$$f(n) = x(n) + x(n+(N/2)),$$

$$g(n) = \{x(n) - x(n+(N/2))\}\ W_N^{n}.$$

In dieser Weise wird die Aufteilung weiter fortgesetzt, wodurch sich schließlich der Sande-Tukey-Algorithmus ergibt. Zwar verläuft die Berechnung des Sande-Tukey-Algorithmus' analog zu der für die Zeitdezimierung, aber die beiden Algorithmen unterscheiden sich in ihrer Philosophie doch deutlich voneinander. Wie gezeigt, liegt der Unterschied darin, daß für die Zeitdezimierung der Laufindex im Objektbereich in gerade und ungerade Werte sowie im Frequenzbereich in eine untere und eine obere Hälfte eingeteilt ist, während für die Frequenzdezimierung die Aufteilung umgekehrt ist. Bild 4.7 zeigt die SFG der beiden Formen des Sande-Tukey-Algorithmus'.

Betrachten wir die Radix-2-Schmetterlinge für die DIT-Form (decimation in time) und die DIF-Form (decimation in frequency), so ergeben sich folgende Grundoperationen:

$$\text{DIT: } X_f(i) = x(i) + W_N^{nk}x(i+2^L), \qquad X_f(i+2^L) = x(i) - W_N^{nk}x(i+2^L),$$

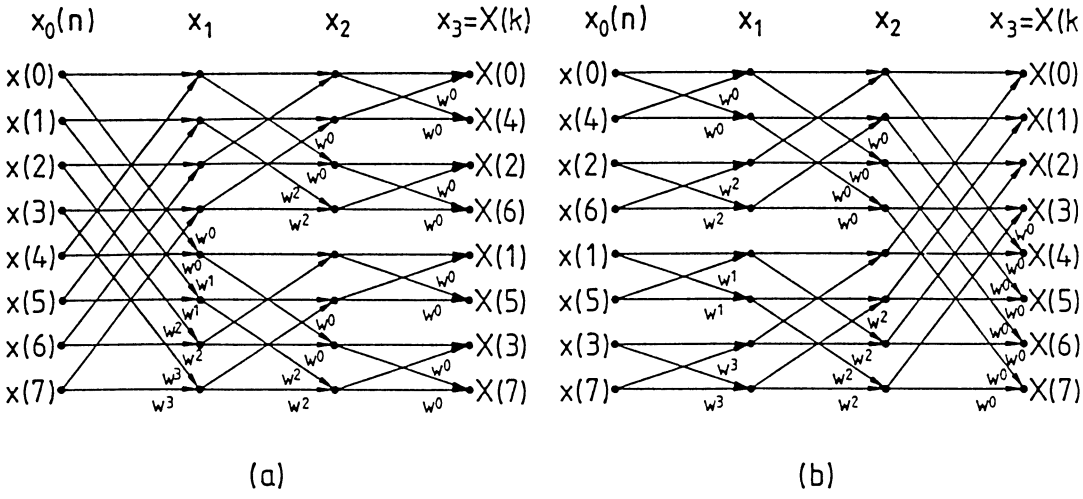


Bild 4.7. Sande–Tukey–Signalflußgraphen: a) Eingangswerte in natürlicher Reihenfolge, b) Eingangswerte in Bitumkehrreihenfolge, nach [2.1]

$$\text{DIF: } X_f(i) = x(i) + x(i+2^L), \qquad X_f(i+2^L) = \{x(i) - x(i+2^L)\}\, W_N^{nk},$$

$$i = 0,1,\ldots,N-1, \qquad n,k = 0,1,\ldots,N-1,$$

$$L = 1,2,\ldots,r, \qquad r = \log_2 N.$$

Wenn man den DIT–Algorithmus ausgehend von Daten in natürlicher Reihenfolge ausführt, muß der Exponent der W_N^k aus den Adress–Bits der niedrigeren Ordnung in BRO erzeugt werden. Dies ist unzweckmäßig. Deshalb geht man beim DIT–Algorithmus i.allg. von Daten in umgeordneter Reihenfolge aus. Für den DIF–Algorithmus gilt genau das umgekehrte.

Bezüglich der Anzahl der auszuführenden Operationen unterscheiden sich die DIT– und die DIF–Formen nicht. Für die Anzahl der komplexen Multiplikationen bzw. Additionen erhält man für die Radix–2–Version [3.9]

$$M_c = (N/2)\log_2 N, \qquad A_c = N \cdot \log_2 N.$$

Eine komplexe Multiplikation kann man entweder durch 4 reelle Multiplikationen und 2 reelle Additionen oder durch 3 reelle Multiplikationen und 3 reelle Additionen ersetzen. Entsprechend erhält man

$$M_r = 2N \cdot \log_2 N, \qquad A_r = 3N \cdot \log_2 N,$$

bzw. $$M_r = (3N/2) \cdot \log_2 N, \qquad A_r = (7N/2) \cdot \log_2 N.$$

Die hier genannten Werte schließen auch triviale Operationen ein (z.B. Multiplikationen mit 1 oder j). Insbesondere für kleine Werte von N mag es vorteilhaft sein, diese auszuschließen. In der Tabelle 4.2 sind die Anzahl der Operationen gemäß den oben angegebenen Ausdrücken mit 4 reellen Multiplikationen und 2 Additionen je komplexer Multiplikation angegeben. Die eingeklammerten Zahlen ergeben sich für den Fall von 3 reellen Multiplikationen und 3 Additionen je komplexer Multiplikation sowie unter Fortlassung trivialer Multiplikationen und optimaler Ausnutzung der Symmetrien trigonometrischer Funktionen.

Tabelle 4.2. Anzahl der reellen Multiplikationen und Additionen zur Berechnung der komplexen FFT der Länge N [3.9], [4.7]

N	Radix-2		Radix-4		Split-Radix	
	Mult.	Add.	Mult.	Add.	Mult.	Add.
16	128	192	36	164	20	148
	(24)	(152)	(20)	(148)		
32	320	480			68	388
	(88)	(408)				
64	768	1152	288	1128	196	964
	(264)	(1032)	(208)	(976)		
128	1792	2688			516	2308
	(721)	(2504)				
256	4096	6144	1728	5824	1284	5380
	(1800)	(5896)	(1392)	(5488)		
512	9216	13824			3076	12292
	(4360)	(13576)				
1024	20480	30720	9216	29696	7172	27652
	(10248)	(30728)	(7856)	(28336)		
2048	45056	67584			16388	61444
	(23560)	(68616)				

4.2.2 Radix-4- und Split-Radix-Algorithmus für die FFT

Wie im Abschnitt 3.4.1 ausgeführt wurde, kann die Indizierung der Datenwerte durch Primfaktoren oder durch Potenzen von nur einer Primzahl erfolgen. In den bisherigen Abschnitten dieses Kapitels haben wir nur die Basis 2 benutzt. Wir wollen nun die Verwendung höherer Radizes betrachten. Dabei interessieren vor allem Potenzen von 2, da diese in der Signalverarbeitung eine besonders wichtige Rolle spielen.

Zuerst befassen wir uns mit der Radix-4-FFT. Hierbei werden jeweils 4 Datenwerte miteinander verknüpft (vgl. Bild 3.4(b)). Beim Vergleich mit einem Radix-2-SFG erkennt man, daß eine Iteration des Radix-4-Algorithmus' zwei Radix-2-Iterationen ersetzt. Das verlangt natürlich eine andere Matrixfaktorisierung und somit andere Drehfaktoren. Bild 4.8 illustriert diesen Sachverhalt. Hierbei ist ein Teilgraph aus dem SFG einer Radix-4-Sande-Tukey-FFT angegeben. Abhängig davon in welcher Position und in welcher Iteration sich dieser Teilgraph befindet, sind in den linken Knoten zusätzliche Drehfakto-

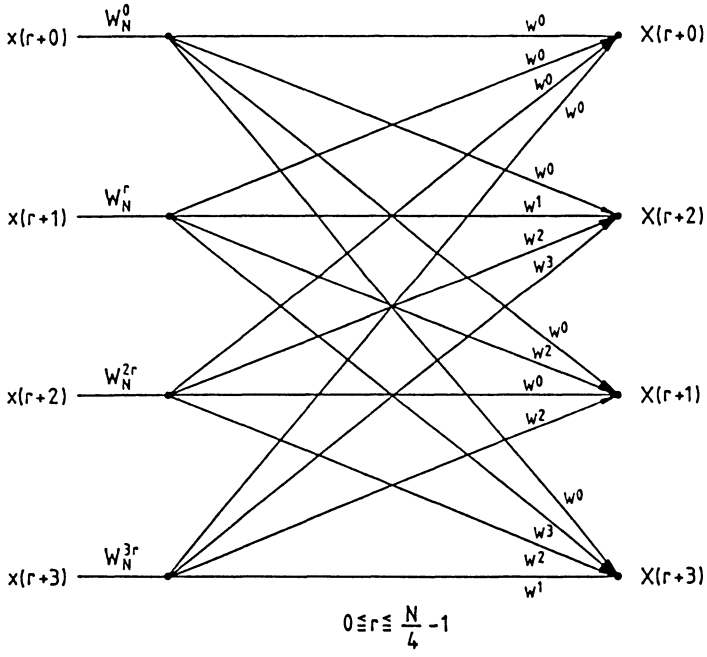


Bild 4.8. Teilgraph einer (DIF-) Radix-4-FFT, nach [1.3]

ren notwendig. Der Vorteil eines Algorithmus' mit höherer Radix besteht in der reduzierten Anzahl von Operationen, insbesondere von Multiplikationen (vgl. hierzu Tabelle 4.2). Als Nachteil ist die Tatsache zu vermerken, daß 4 Variable miteinander zu verknüpfen sind, ehe sie (bei "in-place"-Verarbeitung) wieder zurückgespeichert werden können. Dazu sind verschiedene Zwischenspeicherungen und Aufrufe erforderlich, wodurch der o.g. Vorteil relativiert wird. Die beschriebene Tendenz setzt sich bei höheren Potenzen der Basis 2 fort (Radix-8, Radix-16). Allerdings fallen die Einsparungen an Rechenoperationen immer bescheidener aus, während der Aufwand an Speicherzugriffen stark ansteigt. Deshalb verwendet man in der Praxis kaum Algorithmen mit höherer Basis als 4. Eine Möglichkeit, die Vorteile der Radix-2- und der Radix-4-Verarbeitung miteinander zu kombinieren, bietet der im folgenden beschriebene Algorithmus.

Der Split-Radix-FFT-Algorithmus (SR-FFT) [4.7], [4.8], [4.9] beruht auf einer einfachen Überlegung: Es wird jeweils ein Radix-4-Algorithmus für die ungerade indizierten Werte der DFT benutzt und ein Radix-2-Algorithmus für gerade indizierten Werte. Der Algorithmus basiert auf folgenden Zerlegungen:

Wenn

$$X_f(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}$$

zu berechnen ist, wird dies zerlegt in

$$X_f(2k) = \sum_{n=0}^{(N/2)-1} \{x(n)+x(n+(N/2))\} W_N^{2kn}$$

für die geraden Terme, sowie

$$X_f(4k+1) = \sum_{n=0}^{(N/4)-1} \{[x(n)-x(n+(N/2)] - j[x(n+(N/4))-x(n+(3N/4))]\} W_N^n W_N^{4kn}$$

und

$$X_f(4k+3) = \sum_{n=0}^{(N/4)-1} \{[x(n)-x(n+(N/2)] + j[x(n+(N/4))-x(n+(3N/4))]\} W_N^{3n} W_N^{4kn} \qquad (4.14)$$

für die ungeraden Terme.

Die erste Iteration einer Split-Radix-Zerlegung mit Frequenzdezimierung ersetzt eine N-Punkte-DFT durch eine (N/2)-Punkte-DFT und zwei (N/4)-Punkte-

DFT. Die N-Punkte-DFT wird dann durch wiederholte Anwendung solcher Zerlegungen bis hin zur letzten Iteration ersetzt, für die zusätzlich noch einige der üblichen Radix-2-Schmetterlinge (ohne Drehfaktoren) benötigt werden.

Da man i. allg. beabsichtigt, den Split-Radix-Algorithmus "in-place" für reelle Daten durchzuführen, braucht man eine schematische Darstellung, wie sich der Algorithmus in verschiedenen Iterationen verhält, sowie eine Übersicht über die Speicherplätze in denen die Zwischenergebnisse abgespeichert werden.

Der Flußgraph für den Algorithmus hat die Struktur einer Radix-2-FFT mit Ausnahme der Anordnung der Drehfaktoren. Tatsächlich ist es gerade die Anordnung der Drehfaktoren, die den Algorithmus so wesentlich unterschiedlich von den anderen macht. Der L-förmige SR-FFT-Schmetterling verlegt die Berechnung der obere Hälfte des SFG um eine von r Iterationen nach vorn, während in der unteren Hälfte (ähnlich wie beim Radix-4-Schmetterling) zwei Iterationen als eine berechnet werden. Das Bild 4.9 zeigt dies deutlich.

Im Gegensatz zu anderen Algorithmen, wie z.B. zu den Cooley-Tukey-FFT-Algorithmen oder den Primfaktor-Algorithmen, arbeitet der Split-Radix-FFT-Algorithmus nicht Iteration nach Iteration indexweise ab. Statt dessen berechnet die erste Iteration zwar immer N/2 Schmetterlinge und die zugehörigen Drehfaktoren. Die nächste Iteration operiert aber nur auf N/2 Datenpunkten, weil

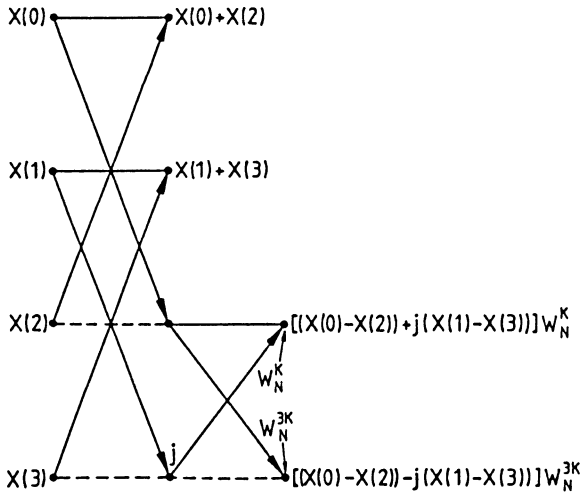


Bild 4.9. Schmetterling des Split-Radix-Algorithmus', nach [4.7]

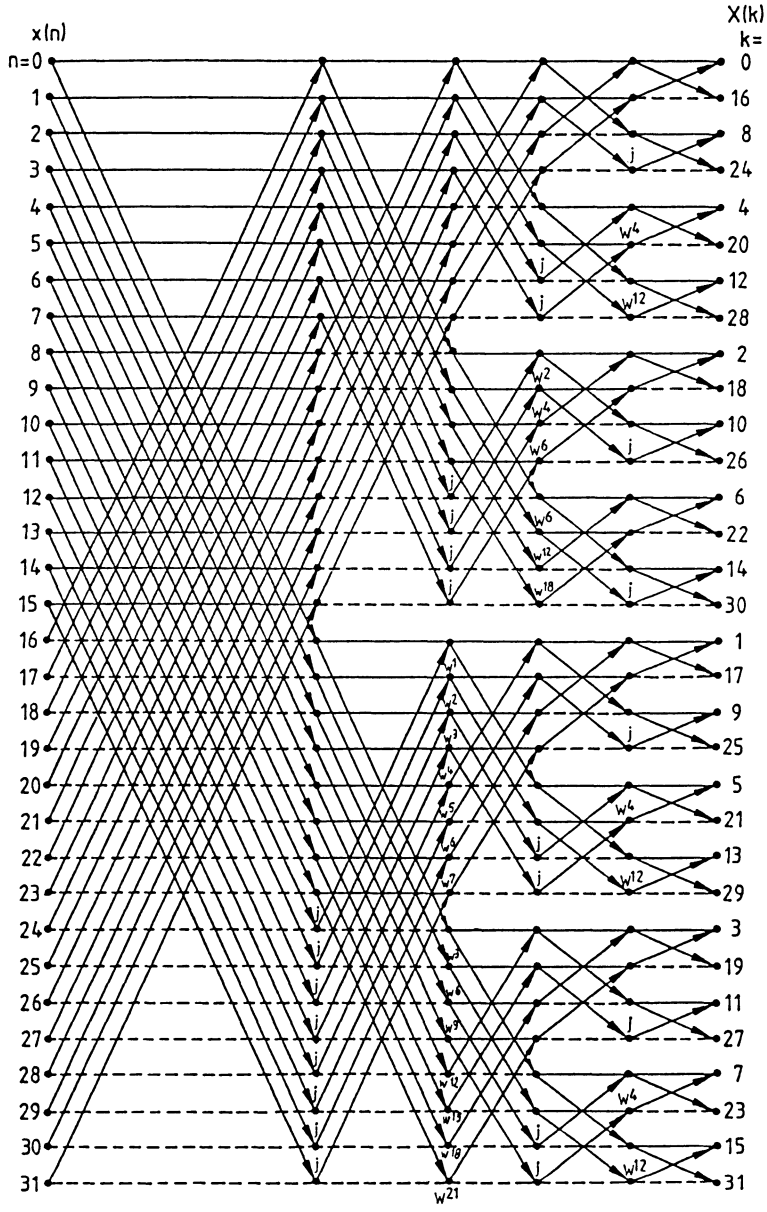


Bild 4.10. Signalflußgraph des Split-Radix-Algorithmus' für N=32 (r=5), nach [4.7]

die L-förmigen Schmetterlinge der ersten Iteration schon die Hälfte der zweiten Iteration berechnet haben. Nur der verbleibende Teil dieser Iteration wird mit der nächsten Iteration berechnet.

Ein FORTRAN-Programm, das eine solche Indizierung benutzt, ist im Anhang von [4.9] angegeben. Als Beispiel eines Split-Radix-SFG illustriert Bild 4.10 eine FFT der Länge N=32.

Stufe 1	Stufe 2	Stufe 3	Stufe 4	Stufe 5
0	0	0	0	
1	2	4	8	
2	4	8		4
3	6	12		12
4	8		2	
5	10		10	
6	12		6	
7	14		14	
8		1	1	
9		5	9	
10		9		5
11		13		13
12		3	3	
13		7	11	
14		11		7
15		15		15

Bild 4.11. Speicherbelegung beim Split-Radix-Algorithmus, nach [4.7]

Wir erläutern nun den Split-Radix-Algorithmus für komplexe Daten, wofür Bild 4.11 die Speicherbelegung einer 16-Punkte-FFT zeigt. Dieser Algorithmus kann wie folgt kommentiert werden: Iteration 1 wendet Schmetterlinge auf einen Block der Länge N an. Dieser Block dient zur Berechnung aller transformierten Datenwerte $X_f(k)$ $(k = 0,1,\ldots,15)$. Nach dem ersten Schritt haben wir nun einen Block der Länge N/2, der zur zweiten Iteration gehört, aus dem die geraden Punkte $X_f(2k)$ $(k = 0,1,\ldots,7)$ berechnet werden, sowie zwei Blöcke der Länge N/4, die zur 3. Iteration gehören und für die Berechnung von $X_f(4k+1)$ $(k = 0,1,\ldots,3)$ benutzt werden. Dies führt zu der L-Form des Schmetterlingsdiagramms (vgl. Bild 4.9), das eine N-Punkte-DFT durch eine (N/2)-Punkte-DFT und zwei (N/4)-Punkte-DFT ersetzt.

Wenn der Split-Radix-Algorithmus für reelle Daten verwendet wird, ist der gesamte Block in Iteration 1 reell. Außerdem wird ein Block der Länge N/2 in der dritten Iteration erzeugt, der ebenfalls reell ist. Die beiden anderen Blöcke der Länge N/4 in der dritten Iteration sind jedoch komplex. $\{X_f(4k+3)\}$ braucht nicht berechnet zu werden, so daß die entsprechenden Speicherplätze im SFG zum Speichern des imaginären Teils bei der Berechnung von $\{X_f(4k+1)\}$ verwendet werden können. Dieser Prozeß wird wiederholt, bis die gesamte Transformation durchgeführt ist.

Wir diskutieren nun die Komplexität des Split-Radix-Algorithmus' für komplexe Daten. Da der Split-Radix-Algorithmus eine Kombination des Radix-2- und des Radix-4-Algorithmus' ist, vermutet man, daß seine arithmetische Komplexität zwischen den Komplexitäten von Radix-2 und Radix-4 liegen wird.

Tatsächlich ist die Anzahl der Multiplikationen und Additionen noch niedriger als die des Radix-4-Algorithmus'.

Wir bezeichnen die Anzahl der zur Ausführung der komplexen 2^r-Punkte-FFT mit dem Split-Radix-Algorithmus benötigten Multiplikationen mit $M_c(r)$ und die der Additionen mit $A_c(r)$. Damit erhält man aus (4.14)

$$M_c(r) = M_c(r-1) + 2 \cdot M_c(r-2) + 3 \cdot (2^{r-1}) - 8.$$

Mit den Anfangsbedingungen $M_c(1)=0$, $M_c(2)=0$ wird

$$M_c(r) = 2^r \cdot (r-3) + 4.$$

Ohne Berücksichtigung der Anzahl der erforderlichen Additionen zur Ausführung der komplexen Multiplikationen ergeben sich die übrigen als $r \cdot 2^{r+1}$, weil ein neuer Punkt in jeder der r Iterationen durch eine komplexe Addition erzeugt wird. Damit ergibt sich

$$A_c(r) = r \cdot 2^{r+1} + M_c(r) = 3 \cdot 2^r \cdot (r-1) + 4,$$

denn die Anzahl der zur Berechnung eines komplexen Produktes benötigten reellen Additionen ist gleich der Anzahl der Multiplikationen. Tabelle 4.2 stellt die Anzahl von Multiplikationen und Additionen für einige bekannte Algorithmen zusammen [4.7].

Es ist bemerkenswert, daß die Anzahl der Drehfaktoren beim Radix-2-, Radix-4- und beim Split-Radix-Algorithmus gleich groß ist. Trotzdem benötigt der Split-Radix-Algorithmus die geringste Anzahl von nicht-trivialen Multiplikationen. Eine Betrachtung der klassischen Algorithmen wie z.B. Radix-2 und Radix-4 zeigt, daß alle diese Algorithmen einschließlich des Split-Radix-Algorithmus' unter Verwendung des gleichen, auf Radix-2-Schmetterlingen basierenden SFG implementiert werden. Daher kann der Split-Radix-Algorithmus als zur gleichen Klasse von Algorithmen gehörig angesehen werden.

Hier stellt sich die Frage, ob es möglich ist, die Komplexität durch die Berechnung der ungeraden Teile der DFT mit einem noch höheren Radix zu verbessern. Verschiedene Zerlegungen sind untersucht worden, jedoch wurde keine gefunden, die besser als der (2/4)-Algorithmus ist [4.7]. Man kann deshalb vermuten, daß die Abarbeitung durch separate Radix-2- und Radix-4-Schmetterlinge für die geraden bzw. die ungeraden Teile der DFT für diese Klasse von Algorithmen die Anzahl der Multiplikationen auf ein Minimum reduziert.

In [4.7] wird die Anzahl der Multiplikationen und Additionen beim Split-Radix-Algorithmus mit denjenigen einiger anderer kürzlich entwickelter Algorithmen [4.10], [4.11], [4.12] verglichen. Da die diskrete Cosinus-Transformation (DCT) als eine Abwandlung der DFT angesehen werden kann, ist der Split-Radix-Algorithmus auch bei der Berechnung der FCT anwendbar.

Wenn der Split-Radix-Algorithmus zur Berechnung der DFT eines reellen Datensatzes verwendet wird, sind $X_f(k)$ und $X_f(N-k)$ konjungiert komplex. Es ist deshalb nutzlos, sowohl $\{X_f(4k+1)\}$ als auch $\{X_f(4k+3)\}$ zu berechnen, weil $X_f(4k+3) = X_f(N-(4k+1)) = X_f^*(4k+1)$ gilt. Diese Beobachtung erlaubt die Berechnung der arithmetischen Komplexität des Split-Radix-Algorithmus' für reelle Daten. Hierbei wandelt man gemäß (4.14) eine reelle DFT der Länge 2^r in eine reelle DFT der Länge 2^{r-1}, um $X_f(2k)$ zu gewinnen, und eine komplexe DFT der Länge 2^{r-2}, um $X_f(4k+1)$ zu gewinnen. Die Kosten betragen

$3 \cdot (2^{r-2}-2)+2$ reelle Multiplikationen

und $3 \cdot (2^{r-2}-2)+2$ reelle Additionen (wegen der Drehfaktoren),

plus 2^r Additionen.

Nun kann man die Anzahl der reellen Multiplikationen $M_r(r)$ und die Anzahl der reellen Additionen $A_r(r)$ einer reellen 2^r-DFT als

$$M_r(r) = 2^{r-1}(r-3) + 2 = M_c(r)/2$$

bestimmen. Die Anzahl der reellen Additionen ermittelt man als

$$A_r(r) = A_r(r-1) + A_c(r-2) + 2^r + 3\cdot 2^{r-2} - 4 = A_r(r-1) + 2^{r-2}\cdot(3r-2).$$

Mit $A_r(1) = 2$ erhält man $A_r(r) = 2^{r-1}\cdot(3r-5) + 4 = A_c(r)/2 - 2^r + 2$.

4.2.3 Primfaktor-Algorithmus für die DFT

Von einem übergeordneten Standpunkt aus betrachtet, handelt es sich bei den im Abschnitt 4.2.1 dargestellten Algorithmen um die einfachsten Sonderfälle eines allgemeineren Prinzips. Das manifestiert sich in der Darstellung der Indizes n und k. Eine Verallgemeinerung von (4.4) lautet für N=4

$$n = N_0 n_1 + N_1 n_0 \qquad k = N_0 k_1 + N_1 k_0 \qquad \text{mit} \qquad N_0 N_1 = N.$$

Im Fall des Cooley-Tukey-Algorithmus' handelt es sich um einen Radix-2-Algorithmus, bei dem $N_0 = 2^1$ und $N_1 = 2^0$ ist.

Beim Primfaktor-Algorithmus sind N_1 und N_2 teilerfremd. Damit kann man Algorithmen für $N \neq p^i$ Abtastwerte konstruieren (p Primzahl, i=1,2,3,...). Der Ausdruck Primfaktor-Algorithmus bezieht sich also auf die Verwendung von teilerfremden (relative prime) Faktoren N_1 für die Darstellung der Indizes n und k (vgl. Abschnitt 3.4.1).

Es gibt zwei Formen von Algorithmen, die diese Darstellung benutzen, nämlich der in diesem Abschnitt beschriebene Primfaktor-Algorithmus (PFA) und eine verschachtelte Version, die Winograd-Fourier-Transformationsalgorithmus (WFTA) heißt und im Abschnitt 4.2.4 behandelt wird.

Aus Gründen der Übersichtlichkeit wird der PFA hier für ein Beispiel mit 2 Faktoren $N_1 \cdot N_2 = N$ entwickelt [3.8]. Für die Indizes im Objekt- und im Frequenzbereich schreiben wir (vgl. Abschnitt 3.4.1)

$$n = < C_1 n_1 + C_2 n_2 >_N \quad \text{und} \quad k = < C_3 k_1 + C_4 k_2 >_N .$$

wobei $<z>_N \equiv z \bmod N$ ist (vgl. Abschnitt 3.2).

Für eine eindeutige Indizierung müssen die Bedingungen

$$C_1 = a \cdot N_2 \quad \text{und} \quad C_2 = b \cdot N_1 \quad \text{sowie} \quad C_3 = c \cdot N_2 \quad \text{und} \quad C_4 = d \cdot N_1$$

erfüllt sein. Dabei sind a und N_1 sowie c und N_1 bzw. b und N_2 sowie d und N_2 teilerfremd (a,b,c,d sind natürliche Zahlen).

Die Bedingungen für die gegenseitige Entkopplung der Module ist (vgl. Abschnitt 3.4.1) $<C_1 C_4>_N = 0$ und $<C_2 C_3>_N = 0$. Eine DFT mit den Summenlängen N_1 und N_2 verlangt außerdem $<C_1 C_3>_N = N_2$ und $<C_2 C_4>_N = N_1$.

Ein Koeffizientensatz, der diese Bedingungen erfüllt, ist [3.8]

$$a = b = 1 \tag{4.15}$$

$$\text{und} \quad C_3 = N_2 <N_2^{-1}>_{N1} \quad \text{und} \quad C_4 = N_1 <N_1^{-1}>_{N2} . \tag{4.16}$$

Dies führt zu einfachen Koeffizienten für die Objektindexbezeichnung und zu etwas komplizierteren für die Frequenzindexbezeichnung. Diese Frequenzindizierung benutzt das Chinesische Rest-Theorem (vgl. Abschnitt 3.2). Das Ergebnis der Auswahl von Koeffizienten in (4.16) führt zu

$$k_1 = <k>_{N1} \quad \text{und} \quad k_2 = <k>_{N2} .$$

Es sind auch andere Indizierungen möglich, jedoch werden bei dieser Wahl C_1 und C_2 als Folge von (4.15) so klein wie möglich. Die beiden Indizierungen gemäß (4.15) und (4.16) ergeben sich dann als

$$n = <N_2 n_1 + N_1 n_2>_N \quad \text{und} \quad k = <C_3 k_1 + C_4 k_2>_N . \tag{4.17}$$

Mit (4.8) wird beispielsweise

$$C_3 = 2 \cdot N_2 \text{ und } C_4 = 2 \cdot N_1 .$$

Für die DFT schreiben wir dann [2]

$$X_f(k_1,k_2) = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x(n_1,n_2)\, W_{N1}^{n1k1}\, W_{N2}^{n2k2}. \qquad (4.18)$$

Gleichung (4.18) illustriert die Tatsache, daß die Methoden der FFT letztlich auf der Überführung in mehrdimensionale Transformationen beruhen. Der PFA mit 2 Faktoren gemäß (4.18) ist eigentlich eine 2D-Transformation der Längen N_1 und N_2. Es ist bemerkenswert, daß durch Benutzung teilerfremder Faktoren N_1 und N_2 eine vollständige Entkopplung der Summationen erreicht wird. Deshalb kann die Reihenfolge der Summen vertauscht werden. Falls die N_i gemeinsame Faktoren enthalten (was natürlich auch bei den Radix-p-Transformationen der Fall ist), entstehen sog. Twiddle-Faktoren. Das sind Multiplikatoren der Form W_N^{nikj} mit $i \neq j$. In diesem Fall ist keine Entkopplung der Summationen gegeben und somit keine Vertauschung möglich.

Als Beispiel eines PFA mit zwei Faktoren betrachten wir $N_1 = 3$ und $N_2 = 5$.

Mit $N = 3 \cdot 5 = 15$ hat man aus (4.17)

$$n = (5n_1 + 3n_2) \bmod 15 \quad \text{und} \quad k = (10k_1 + 16k_2) \bmod 15,$$

wobei $n_1, k_1 = 0,1,2$ und $n_2, k_2 = 0,1,2,3,4$.

Damit erhält man die zweidimensionale Abbildung der Indizes wie in Tabelle 4.3 auflistet. Eine symbolische Darstellung der beiden entkoppelten Module zeigt das Bild 4.12.

Wenn die Länge N durch drei teilerfremde Faktoren $N = N_1N_2N_3$ gegeben ist, wird die Objektindexbezeichnung

$$n = < N_2N_3n_1 + N_1N_3n_2 + N_1N_2n_3 >_N.$$

---

[2] Aus drucktechnischen Gründen werden Indizes der Exponenten auf der (hochgestellten) Zeile geschrieben.

Tabelle 4.3. PFA-Indexdiagramme für $N_1 = 3$ und $N_2 = 5$

n_1 \ n_2	0	1	2	3	4	
0	0	3	6	9	12	mod 15
1	5	8	11	14	2	
2	10	13	1	4	7	

n_1 \ n_2	0	1	2	3	4	
0	0	6	12	3	9	mod 15
1	10	1	7	13	4	
2	5	11	2	8	14	

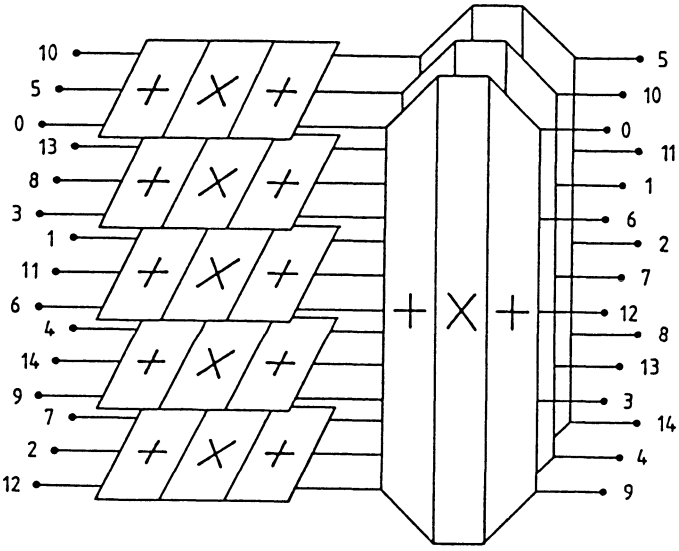


Bild 4.12. Module der Primfaktor-FFT mit zwei Faktoren, nach [3.8]

Die zugehörige DFT wird dann wie folgt geschrieben

$$X_f(k_1,k_2,k_3) = \sum_{n_3=0}^{N_3-1} \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x(n_1,n_2,n_3)\, W_{N1}^{n1k1}\, W_{N2}^{n2k2}\, W_{N3}^{n3k3}.$$

Vier Faktoren würden vier verschachtelte Summationen ergeben, etc. Die Summenbildung kann in beliebiger Reihenfolge aufgeführt werden.

Wie die oben ausgeführten Beispiele zeigen, benötigt man für die Implementierung eines PFA i.allg. mehrere "kurze" Module, deren Längen den Primfaktoren N_i entsprechen. Derartige kurze Module sind in der Literatur angegeben (z.B. [2.2]). Wir zeigen hierzu ein Beispiel, ohne jedoch eine Liste der kurzen Module anzugeben. Es sei N=12, dann bieten sich die Module mit N_1=3 und N_2=4 an. (Korrigierte) Algorithmen aus [2.2] ergeben die SFG gemäß Bild 4.13. Ihre Kombination für N=12 ist im Bild 4.14 zu sehen.

Wie die Bilder 4.13 und 4.14 deutlich machen, ist auch beim PFA eine "in-place"-Berechnung möglich, allerdings unter Aufwand von erheblich mehr (schwach besetzten) Iterationen als beim Radix-2-Algorithmus.

Die Anzahl der reellen Multiplikationen und Additionen für den Primfaktor-Algorithmus läßt sich aus den entsprechenden Zahlen der kurzen Module (s.

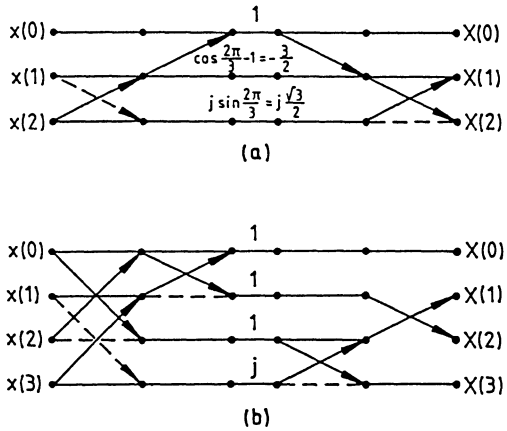


Bild 4.13. PFA-Module: a) N=3, b) N=4, nach [2.2]

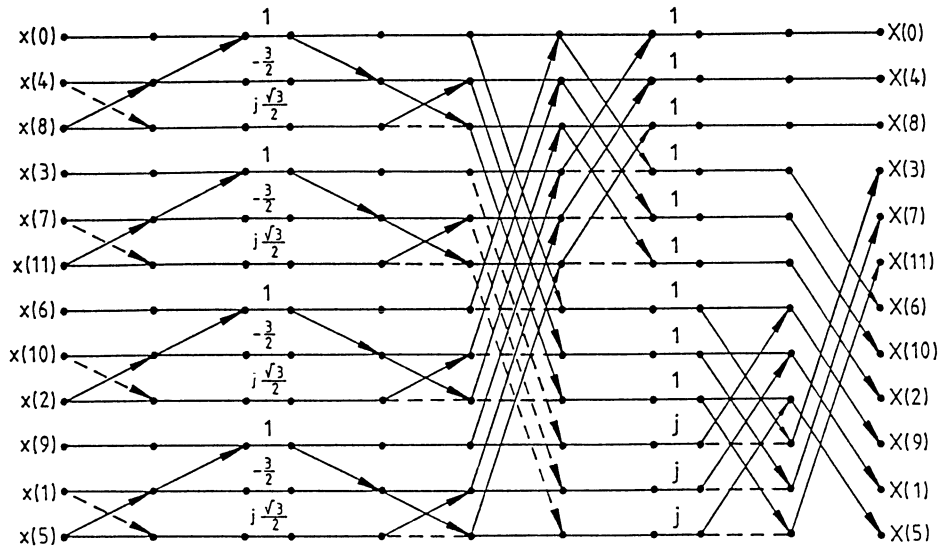


Bild 4.14. PFA-SFG für N=12, nach [2.2]

Tabelle 4.4. Anzahl der reellen Multiplikationen und Additionen für PFA-Module der Länge N von reellen Daten (für komplexe Daten zu verdoppeln) [3.8]

N	2	3	4	5	8	11	16
Multiplikationen	0	2	0	5	2	20	10
Additionen	2	6	8	17	26	84	74

Tabelle 4.5) berechnen. In der ersten Iteration eines PFA mit vier Primfaktoren gibt es $N_4N_3N_2$ DFT mit der Länge N_1, die die $(N_4N_3N_2)$-fache Anzahl von Additionen und Multiplikationen einer DFT der Länge N_1 aus Tabelle 4.4 haben. Hinzu kommen die entsprechenden Anzahlen für die anderen drei DFT. Einige Beispiele (für komplexe Daten) sind in Tabelle 4.5 angegeben. Die Tabelle zeigt, daß der PFA bei geeigneter Wahl der Module gegenüber den Radix-p-Algorithmen beträchtliche Einsparungen an Multiplikationen erlaubt.

Tabelle 4.5. Anzahl der reellen Operationen für eine komplexe Vier-Primfaktor-FFT [3.8]

$N = N_1 \times N_2 \times N_3 \times N_4$	Multiplikationen	Additionen
6 = 2 3 1 1	8	36
12 = 4 3 1 1	16	96
20 = 4 1 1 5	40	216
30 = 2 3 1 5	100	384
63 = 1 7 9 1	284	1236
252 = 4 9 7 1	1136	5952
504 = 8 9 7 1	2524	13164
1008 =16 9 7 1	5804	29100
2520 = 8 9 7 5	17660	82956

4.2.4 Winograd-Algorithmus

Im Jahre 1976 veröffentlichte Winograd einen neuen Algorithmus zur effizienten Berechnung der DFT, der erheblich weniger Multiplikationen als die traditionelle Cooley-Tukey-FFT benötigt, jedoch unter Aufwand von mehr Additionen. Der Algorithmus verwendet die Primfaktor-Indexbezeichnung für das Indizieren und benutzt die zyklische Faltung für die kurzen DFT-Module. Die Operationen werden so umgeordnet, daß die Multiplikationen vom Algorithmus zwecks Reduktion ihrer Anzahl zusammengefaßt werden können. Der Winograd-Fourier-Transformationsalgorithmus (WFTA) ähnelt dem oben beschriebenen PFA, jedoch mit Verschachteln der Multiplikationen.

Um den WFTA wirksam zu benutzen, ist es notwendig, mehr von der Theorie dieses schnellen DFT-Algorithmus' zu verstehen als es für den PFA nötig war [3.8]. Die Idee ist, aus dem Datenvektor der Länge N durch einfache Additionen eine Menge von $M \geq N$ Zwischenvariablen zu bilden. Diese werden dann mit Konstanten multipliziert, die aus einer Kombination des reellen und des imagnären Teils der W_N in der DFT abgeleitet werden. Diese N Produkte werden schließlich durch einfache Additionen kombiniert, um die N Werte der DFT zu erzeugen. Man beachte die Folge von Additionen, Multiplikationen und Addi-

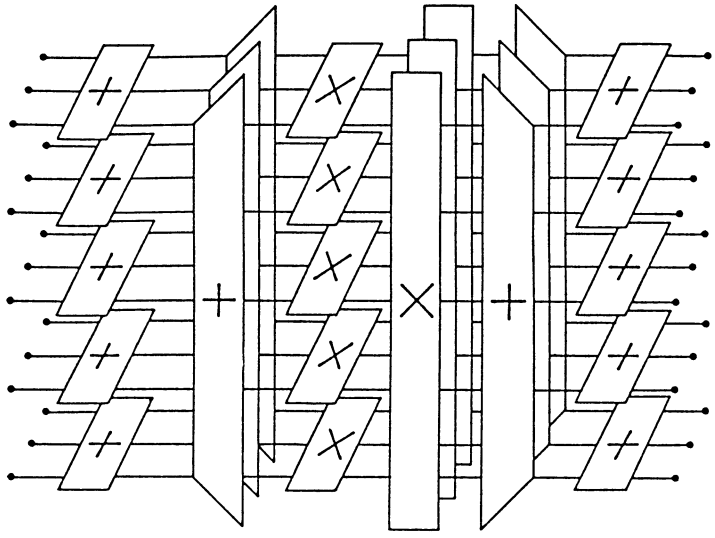


Bild 4.15. Module des WFTA mit verschachtelten Multiplikationen, nach [3.8]

tionen im Bild 4.12, sowie die Aufblähung der Zwischenvariablen (M > N). Diese Strategie wird auch als die Kolba-Parks-Version des WFTA bezeichnet [4.13].

Der Weg, eine DFT über eine Faltung zu ermitteln, erscheint zunächst ungewöhnlich. Es kann jedoch sogar zweckmäßig sein, diese Faltung ihrerseits über eine DFT (unterschiedlicher Länge!) zu berechnen. Für die Technik hierzu verweisen wir auf die einschlägige Literatur [2.2], [3.9] und [4.14]. Die Algorithmen erfordern die Berechnung einer Transformationsmatrix [T]

$$\underline{X}_f = [T]\cdot\underline{x} = [C][B][A]\cdot\underline{x},$$

wobei [C] eine (N×M)-Matrix, [A] eine (M×N)-Matrix und [B] eine (M×M)-Diagonalmatrix ist. In den o.g. Büchern (z.B. [3.9]) findet man diese Matrizen für praktisch interessant Werte von N.

Der WFTA benutzt die gleiche Indexbezeichnung wie der PFA. Die Länge der Module N sind Primzahlen oder Potenzen von Primzahlen. Der WFTA ist deshalb gut geeignet wenn N prim ist. Diese sog. "kurzen" Module können zu "p-dimensionalen" Transformationen der Länge

$$N = \prod_{i=1}^{p} N_i$$

zusammengesetzt werden, wobei durch Verschachteln nochmals Multiplikationen gespart werden.

Die entkoppelte DFT-Form mit 2 Modulen ist

$$X_f(k_1,k_2) = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x(n_1,n_2)\, W_{N1}^{n1k1}\, W_{N2}^{n2k2}.$$

Wir weisen darauf hin, daß die innere Summe eine Menge identischer DFT der Länge N_1 entlang den N_2 Zeilen einer Anordnung $x(n_1,n_2)$ ist (vgl. auch Abschnitt 5.2). Die äußere Summe ist eine Menge von DFT mit den Längen N_2 entlang den N_1 Spalten einer Anordnung von Zwischenergebnissen, die eine Funktion von (k_1,n_2) sind. Von Winograd wurde gezeigt, daß diese Operationen in beliebiger Reihenfolge durchgeführt werden können. Wenn die kurzen DFT-Module in drei sequentiell anzuwendende Operatoren faktorisiert werden, benötigen sie einen Additionsoperator, anschließend einen Multiplikationsoperator und zuletzt einen weiteren Additionsoperator. Diese Operationenfolge ist im Bild 4.12 illustriert. Der breitere Teil in der Mitte der Blöcke deutet an, daß die Anzahl der Multiplikationen im allgemein größer als die Anzahl der Datenpunkte ist. Unter Verwendung der Möglichkeit zur Umordnung von Operationen können die Multiplikationsetappen benachbart angeordnet werden, wie es im Bild 4.15 beispielhaft gezeigt ist.

Bei der Durchführung werden die Variablen nicht (wie sonst üblich) zwischen zwei Iterationen zuerst mit einer Konstanten und dann mit anderen Variablen multipliziert. Stattdessen werden die beiden Multiplikations-Stufen in eine Stufe von Multiplikationen der Variablen mit Konstanten zusammengefaßt. Die Multiplikationen der Variablen konzentrieren sich in der Mitte des Algorithmus'. Diesen Vorgang nennt Winograd "Verschachteln" (nesting). Er ist entscheidend für die geringe Anzahl von Multiplikationen im WFTA.

Man beachte, daß die Struktur des Additionsoperators eine Aufblähung von Daten in der Mitte des Algorithmus' verursacht. Diese Aufblähung ist ein Hauptproblem bei der Ausführung des zentralisierten WFTA, da sie eine di-

rekte Operation mit Ersetzung ("in-place") verhindert und eine Zunahme der Anzahl der Additionen und der abzuspeichernden multiplikativen Konstanten zur Folge hat.

Die Anzahl der Multiplikationen und Additionen ist aus der Tabelle 4.6 zu ersehen. Sie unterscheidet sich von Tabelle 4.4 für den PFA umso mehr, je größer N wird. Der WFTA erlaubt die gleichen Transformations-Längen N wie der PFA, aber die Anzahl von Multiplikationen ist kleiner und die Anzahl der Additionen größer. Die Wirksamkeit des PFA hängt von der der individuellen Module ab. Es muß nochmal betont werden, daß eine einfache Zählung von Operationen kein Gesamtbild für die Auswertung von Algorithmen ergibt. Der WFTA benötigt mehr Indexberechnungen als der PFA. Die Datenaufblähung verhindert die "in-place"-Berechnung und die Zentralisierung der Multiplikationen verursacht eine viel größere Anzahl von abzuspeichernden Multiplikationsfaktoren als für den PFA.

Bei modernen Vektorrechnern (z.B. Cray und IBM 3090) werden Multiplikationen überlappend mit Additionen ausgeführt und benötigen keinen zusätzlichen Zeitaufwand. Da der WFTA keine Kommutativität erfordert, kann die Transformationsmatrix in Blöcke aufgeteilt und parallel bearbeitet werden. Dadurch kann die Gesamtzahl der Operationen reduziert werden.

Tabelle 4.6. Anzahl der reellen Operationen für einen komplexen Vier-Faktor-WFTA [3.8]

$N = N_1 \times N_2 \times N_3 \times N_4$	Multiplikationen	Additionen
30 = 2 3 1 5	68	384
63 = 1 9 7 1	196	1394
126 = 2 9 7 1	392	3040
252 = 4 9 7 1	784	6584
504 = 8 9 7 1	1572	14428
1008 =18 9 7 1	3548	34416
2520 = 8 9 7 5	9492	99068

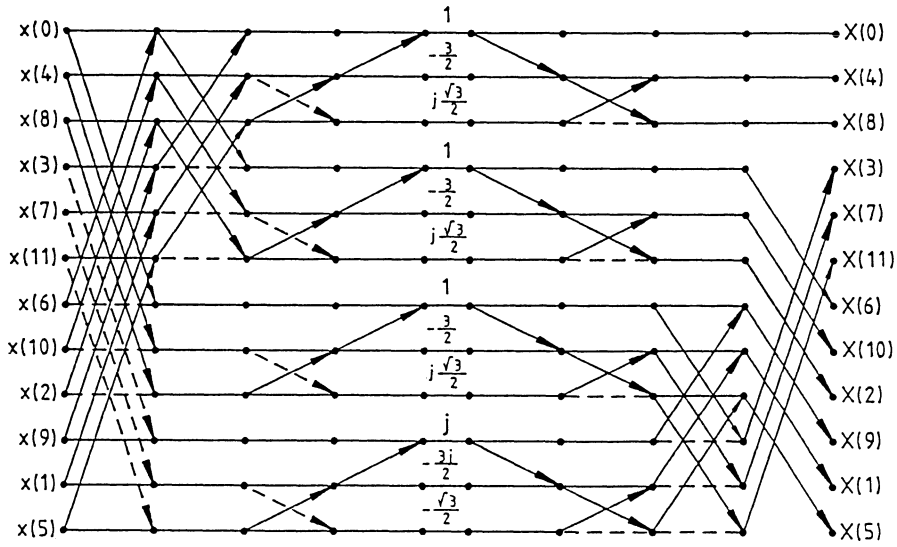


Bild 4.16. WFTA für N=12, nach [2.2]

Die Effizienz des WFTA beruht auf zwei Maßnahmen: (1) Einer Verschachtelung, derart, daß Multiplikationen zusammengefaßt werden können, und (2) einer Berechnung der Module mit Hilfe der zyklischen Faltung und des CRT. Wir verweisen hierzu auf [2.2], [3.9] und [4.14] bis [4.16]. Um die Verschachtelung der Multiplikationen zu demonstrieren, betrachten wir eine FFT mit $N = N_1 \cdot N_2 = 3 \cdot 4 = 12$ Datenwerten unter Verwendung der SFG von Bild 4.13 und Bild 4.14. Wie der im Bild 4.16 dargestellte SFG zeigt, kann man durch Umordnung und Zusammenfassung die Anzahl der nicht-trivialen Multiplikationen von 12 auf 8 reduzieren. Es sei jedoch nochmals betont, daß sich die Effizienz des WFTA erst für größere Transformationslängen und in Verbindung mit den anderen genannten Maßnahmen bemerkbar macht.

4.2.5 Berechnung der DFT aus Walsh-Koeffizienten

In diesem Abschnitt stellen wir einen Algorithmus vor, mit dessen Hilfe die DFT-Koeffizienten aus Walsh-Koeffizienten berechnet werden können [4.17], [3.2]. Dieses Verfahren der Walsh-Fourier-Transformation (W-FT) ist zweckmäßig, wenn die Walsh-Koeffizienten bereits bekannt sind, bzw. wenn nur wenige Fourier-Koeffizienten zu berechnen sind.

Die diskreten Walsh-Funktionen wurden im Abschnitt 2.4 eingeführt und mit Wal(k,n) bezeichnet. Wie dort dargestellt wurde, können die Walsh-Funktionen in gerade (Cal-) und ungerade (Sal-) Funktionen eingeteilt werden. Mit den nach Sequenzen geordneten Walsh-Funktionen hängen diese wie folgt zusammen

$$\mathrm{Wal}_w(2s-1,n) = \mathrm{Sal}(s,n), \qquad \mathrm{Wal}_w(2s,n) = \mathrm{Cal}(s,n),$$

wobei s die Sequenz bezeichnet. Das Bild 2.7 stellt die Menge der ersten acht Walsh-Funktionen in Vergleich mit den entsprechenden Sinus- und Cosinus-Funktionen dar. Eine Betrachtung dieses Bildes zeigt, daß es zu jeder Walsh-Funktion einer gewissen Sequenz eine trigonometrische Funktion mit einer entsprechenden Frequenz gibt.

Wir betrachten zunächst ein einführendes Beispiel mit N=8. Wenn die Sequenzordnung der diskreten Walsh-Funktionen benutzt wird, läßt sich die Walsh-Matrix $[H_w(3)]$ in der folgenden Form repräsentieren (vgl. Abschnitt 2.4):

$$[H_w(3)] = \begin{bmatrix} \mathrm{Wal}(0,n) \\ \mathrm{Sal}(1,n) \\ \mathrm{Cal}(1,n) \\ \mathrm{Sal}(2,n) \\ \\ \mathrm{Cal}(2,n) \\ \mathrm{Sal}(3,n) \\ \mathrm{Cal}(3,n) \\ \mathrm{Sal}(4,n) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ & & & & & & & \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}. \qquad (4.19)$$

Für ein Signal $x(\vartheta)$ können die Abtastwerte durch Auswahl eines ersten Abtastpunktes $\vartheta_0 = \pi/8$ und des Abtastintervalls $2\pi/8$ erzeugt werden [4.17]

$$x(\vartheta_i) = a_0 + \sum_{k=1}^{3} [a_k \cos(k\vartheta_i) + b_k \sin(k\vartheta_i)] + b_4 \sin(4\vartheta_i),$$

mit $\vartheta_i = (4\pi i/8) - (\pi/8)$, und $i = 1,2,\ldots,8$.

Wenn $\underline{x}_8$ ein Spaltenvektor der Ordnung 8 ist, der die Abtastwerte enthält, kann die DWHT der abgetasteten Daten als

$$\underline{X}_{W8} = [H_W(3)]\ \underline{x}_8 \tag{4.20}$$

dargestellt werden.

Die Substitution (4.19) in (4.20) ergibt

$$\underline{X}_{W8} = \begin{bmatrix} X_W(0) \\ X_W(1) \\ X_W(2) \\ X_W(3) \\ \\ X_W(4) \\ X_W(5) \\ X_W(6) \\ X_W(7) \end{bmatrix} = \begin{bmatrix} a_0 \\ Y_1 b_1 + Y_2 b_3 \\ Y_1 a_1 - Y_2 a_3 \\ (1/\sqrt{2}) b_2 \\ \\ (1/\sqrt{2}) a_2 \\ -Y_2 b_1 + Y_1 a_3 \\ Y_2 a_1 + Y_1 a_3 \\ b_4 \end{bmatrix} \tag{4.21}$$

mit $Y_1 = (1/4)\{\cos(\vartheta_0) + \sin(\vartheta_0) + \sqrt{2}\ \cos(\vartheta_0)\}$.

$Y_2 = (1/4)\{\cos(\vartheta_0) - \sin(\vartheta_0) + \sqrt{2}\ \sin(\vartheta_0)\}$.

In (4.21) sind die Fourier-Koeffizienten a_k, b_k unbekannt. Um a_k und b_k zu bestimmen, schreibt man (4.21) um:

$$\begin{bmatrix} a_0 \\ b_1 \\ a_1 \\ b_2 \\ \\ a_2 \\ b_3 \\ a_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} X_W(0) \\ 1{,}30656\ X_W(1) - 0{,}54120\ X_W(5) \\ 1{,}30656\ X_W(2) + 0{,}54120\ X_W(6) \\ 1{,}41421\ X_W(3) \\ \\ 1{,}41421\ X_W(4) \\ 0{,}54120\ X_W(1) + 1{,}30656\ X_W(5) \\ -0{,}54120\ X_W(2) + 1{,}30656\ X_W(6) \\ X_W(7) \end{bmatrix}. \tag{4.22}$$

Gleichung (4.22) zeigt, wie die Fourier-Koeffizienten aus den Walsh-Koeffizienten berechnet werden können. Zuerst werden die Walsh-Koeffizienten $X_W(k)$ durch die DWHT berechnet und dann werden die Fourier-Koeffizienten a_k und b_k durch Substitution von $X_W(k)$ in (4.22) gewonnen.

Wir betrachten nun den allgemeinen Fall mit $N = 2^r$. Die Abtastwerte $x(\vartheta_i)$ können wie folgt beschrieben werden. Aus ihnen ist dann ein Spaltenvektor $\underline{x}_N$ für die Abtastwerte zu erzeugen:

$$x(\vartheta_i) = a_0 + \{ \sum_{k=1}^{(N/2)-1} (a_k \cos(k\vartheta_i) + b_k \sin(k\vartheta_i)) \} + b_{N/2} \sin(\vartheta_i N/2),$$

mit $\vartheta_i = (4\pi i/N) - \pi/N$ und $i = 1,2,\ldots,N$.

Die Walsh-Matrix $[H_w(L)]$ kann durch geeignete Abtastung der Funktionen $wal_w(k,t)$ erzeugt werden (vgl. Abschnitt 2.4). Aus $\underline{x}_N$ und $[H_w(L)]$ kann der Vektor der Walsh-Koeffizienten $X_w(k)$ berechnet werden

$$\underline{X}_{wN} = [H_w(L)]\ \underline{x}_N. \tag{4.23}$$

Die Fourier-Koeffizienten a_k und b_k folgen dann aus (4.22).

Wie im vorstehenden Beispiel für N=8 gezeigt wurde, kann dies durch Lösen eines Gleichungssystems mit N Unbekannten erfolgen. Ein solches Verfahren wäre jedoch für größere N ausgesprochen rechenaufwendig. In [4.17] und [3.2] wird gezeigt, wie die Fourier-Koeffizienten als gewichtete Summen von Walsh-Koeffizienten berechnet werden können. Wir verzichten hier auf die Herleitung der Ausdrücke für die Konvertierungsfaktoren und geben stattdessen ein Beispiel an.

Zu berechnen seien die a_k und b_k mit $k = 1,2,\ldots,(N/2)-1$. Hierzu schreibt man beide Größen als gewichtete Summen der Walsh-Koeffizienten $X_w(k)$:

$$a_k = \sum_i C(k,i) X_w(k), \qquad b_k = \sum_i S(k,i) X_w(k).$$

Dabei wird k als Produkt einer Potenz 2^{j-1} und einer ungeraden Zahl (2q-1) dargestellt: $k = 2^{j-1}(2q-1)$, mit $j, q = 1,2,\ldots$. Die natürliche Zahl $1 \leq i \leq z(k)$ ist begrenzt durch $z(k) = N/2^{j(k)+1}$. Der Zusammenhang zwischen den natürlichen Zahlen k,j und q ist für $k \leq 16$ in Tabelle 4.7 angegeben. Für vorgegebenes k findet man also j und z. Die Gewichtsfaktoren C(k,i) und S(k,i) ermittelt man dann aus

Tabelle 4.7. Beziehung zwischen k, q und j: $k = 2^{j-1}(2q-1)$ [4.17]

k	q	j	k	q	j
1	1	1	9	5	1
2	1	2	10	3	2
3	2	1	11	6	1
4	1	3	12	2	3
5	3	1	13	7	1
6	2	2	14	4	2
7	4	1	15	8	1
8	1	4	16	1	5

$$C(k,i) = (2^{j(k)+2})/N\ [H_{z(k)}]_i \begin{bmatrix} \cos(k\vartheta_{z(k)}) \\ \cos(k\vartheta_{z(k)-1}) \\ \vdots \\ \cos(k\vartheta_1) \end{bmatrix}$$

und

$$S(k,i) = (2^{j(k)+2})/N\ [H_{z(k)}]_i \begin{bmatrix} \sin(k\vartheta_{z(k)}) \\ \sin(k\vartheta_{z(k)-1}) \\ \vdots \\ \sin(k\vartheta_1) \end{bmatrix}.$$

Dabei bezeichnet $[H_{z(k)}]_i$ die i-te Zeile von $[H_{Wz(k)}]$.

Mit diesen Konvertierungsfaktoren können die Fourier-Koeffizienten als gewichtete Summen der Walsh-Koeffizienten angegeben werden:

$$a_k = \sum_{i=1}^{z(k)} C(k,i) X_W\ (i \cdot 2^{j(k)+1} - 2^{j(k)}),$$

$$b_k = \sum_{i=1}^{z(k)} S(k,i) X_W\ (i \cdot 2^{j(k)+1} - 2^{j(k)} - 1).$$

Als Beispiel zeigt Tabelle 4.8 die Werte des Faktors S(k,i) für $N=2^6$. Man beobachtet, daß einige der Faktoren sehr klein sind. Nimmt man sie zu Null an, kann der Rechenaufwand auf Kosten der Genauigkeit weiter vermindert werden.

Tabelle 4.8. Werte der Faktoren S(k,i) für $N=2^6$ und $1 \leq k \leq (N/4)$ [4.17]

k | i =	1	2	3	4	5	6	7	8
1	1,274	-0,528	-0,105	-0,253	-0,025	0,010	-0,052	-0,125
2	1,276	-0,528	-0,105	-0,254	-0,025	0,010	-0,052	-0,126
3	0,426	1,028	-0,687	0,285	-0,086	-0,208	-0,312	-0,129
4	1,281	-0,531	-0,106	0,255				
5	0,256	0,621	0,929	-0,385	-0,206	-0,497	0,332	-0,137
6	0,431	1,040	-0,695	0,288	-0,087	-0,211	-0,315	0,131
7	0,186	-0,077	0,386	0,933	-0,765	0,317	0,063	0,152
8	1,307	-0,541						
9	0,146	-0,061	0,304	0,735	0,895	-0,317	-0,074	-0,178
10	0,265	0,640	0,958	-0,397	-0,212	-0,512	0,342	-0,141
11	0,122	0,294	0,439	-0,182	0,340	0,821	-0,549	0,227
12	0,450	1,086	-0,726	0,301				
13	0,105	0,253	-0,169	0,070	0,231	0,558	0,835	-0,346
14	0,197	-0,816	0,410	0,991	-0,813	0,337	0,067	0,162
15	0,093	-0,039	-0,008	-0,019	0,188	-0,078	0,391	0,945
16	1,414							

k | i =	9	10	11	12	13	14	15	16
1	-0,062	-0,026	0,005	-0,012	0,012	-0,052	-0,026	-0,063
3	-0,019	-0,046	0,031	-0,013	0,042	-0,102	-0,153	0,063
5	-0,034	-0,083	-0,124	0,052	-0,096	-0,233	0,156	-0,644
7	-0,054	0,023	-0,113	-0,274	-0,334	0,138	0,028	0,066
9	-0,084	0,035	-0,175	-0,424	0,348	-0,144	-0,029	-0,069
11	-0,136	-0,329	-0,493	0,204	0,109	0,263	-0,176	0,073
13	-0,257	-0,619	0,414	-0,171	0,052	0,126	0,188	-0,178
15	-0,856	0,355	0,071	0,170	0,017	-0,007	0,035	0,084

Wenn alle Fourier-Koeffizienten gewünscht werden, ist die Anzahl der von der W-FT benötigten Multiplikationen gegeben durch

$$M_r = (4/2)(4/4) + (8/2)(8/4) + \ldots + (N/2)(N/4) = (N^2-4)/6. \qquad (4.24)$$

Für den Fall, daß nicht alle Koeffizienten benötigt werden und die Anzahl der Fourier-Koeffizienten

$$Q = 2^r < N \text{ mit } r = 1,2,\ldots$$

ist, ergibt sich die Anzahl der Multiplikationen als

$$M_r = (Q/2)(N/2^2) + (Q/2^2)(N/2^3) + \ldots + (Q/2^r)(N/2^{r+1}) = N(Q^2-1)/(6Q).$$

Für $N \leq 64$, benötigt die W-FT weniger Multiplikationen als die FFT, für $N>64$ ist die FFT besser. Die W-FT benötigt weniger Multiplikationen für alle N, wenn die Zahl Q der zu berechnenden Koeffizienten relativ klein im Vergleich zu N ist. Der W-FT-Algorithmus ist natürlich dann besonders zweckmäßig, wenn sowohl die Walsh-Koeffizienten wie die Fourier-Koeffizienten zu berechnen sind.

4.2.6 Berechnung der DFT aus Koeffizienten von Teilfolgen

Unter Verwendung der $(WHT)_H$ bei der Berechnung der DFT betrachten wir eine Möglichkeit zur Gewinnung der DFT-Koeffizienten höherer Ordnung aus Koeffizienten niedrigerer Ordnung.

In Matrixdarstellung gilt für die beiden Transformationen:

DFT: $\underline{X}_f = [W_N^E]\ \underline{x}, \qquad L = 1,2,\ldots,r$

$(WHT)_H$: $\underline{X}_{WH} = [H_H(L)]\ \underline{x}, \qquad N = 2^r$

Um die DFT durch die DWHT in Matrixform zu berechnen, schreiben wir

$$\underline{X}_f = N^{-1}\ [W_N^E][H_H(L)]\ \underline{X}_W = [T(L)]\ \underline{X}_{WH}$$

mit der (N×N)-Konversionsmatrix $[T(L)] = N^{-1}[W_N^E][H_H(L)]$.

Um die Beschreibung zu vereinfachen, zerlegen wir jetzt die Matrix $[T_k]$ in N Zeilenvektoren $[T_k]^T = \{\underline{T}_0\ \underline{T}_1\ \ldots\ \underline{T}_k\ \ldots\ \underline{T}_{N-1}\}$. Dabei ist k der Index der Zeilenvektoren von 0 bis N-1. Dann führen wir in dieser Matrix $[T_k]$ mit den Indizes der Zeilenvektoren die Bitumkehroperation durch, z.B. für N=4

$$[T_k] = \begin{bmatrix} \underline{T}_0 \\ \underline{T}_1 \\ \\ \underline{T}_2 \\ \underline{T}_3 \end{bmatrix} \text{ — BRO —> } [T_{<k>}] = \begin{bmatrix} \underline{T}_0 \\ \underline{T}_2 \\ \\ \underline{T}_1 \\ \underline{T}_3 \end{bmatrix}.$$

Die Konversionsmatrix $[T_{<k>}(L)]$ ist orthogonal und hat eine blockdiagonale Struktur, z.B.für N=8

$$[T_{<k>}] = \begin{bmatrix} \underline{T}_0 \\ \underline{T}_4 \\ \underline{T}_2 \\ \underline{T}_6 \\ \\ \underline{T}_1 \\ \underline{T}_5 \\ \underline{T}_3 \\ \underline{T}_7 \end{bmatrix} = \left[\begin{array}{c|c} & \\ [T_{<k>}(L-1)] & 0 \\ & \\ & \\ \hline & \\ & \\ 0 & [R_{<k>}(L-1)] \\ & \end{array}\right]. \qquad (4.25)$$

Dabei ist $[T_{<k>}(L-1)]$ eine (N/2)×(N/2)-Konversionsmatrix und $[R_{<k>}(L-1)]$ eine (N/2)×(N/2)-Restmatrix. Dann ist

$$\underline{X}_{f<k>} = [T_{<k>}(L)]\ \underline{X}_{WHk}. \qquad (4.26)$$

wobei <k> die Bitumkehr von k symbolisiert. Nach der Substitution (4.25) in (4.26) haben wir

$$\underline{X}_{f<k>}(L) = \left[\begin{array}{c|c} [T_{<k>}(L-1)] & 0 \\ \hline 0 & [R_{<k>}(L-1)] \end{array}\right] \cdot \underline{X}_{WHk}. \qquad (4.27)$$

Wir zerlegen nun den Vektor $\underline{x}$ der Ordnung N in zwei Teilvektoren $\underline{x}_1$ und $\underline{x}_2$. Dabei ist $\underline{x}_1$ ein Vektor der Ordnung N/2, der aus den (N/2) oberen Elementen des Vektors $\underline{x}$ besteht, während $\underline{x}_2$ die (N/2) unteren Elementen desselben Vektors enthält. Nun kann $\underline{X}_{WH}$ wie folgt geschrieben werden

$$\underline{X}_{WH}(L) = \begin{bmatrix} [H_H(L-1)] & [H_H(L-1)] \\ \hline [H_H(L-1)] & -[H_H(L-1)] \end{bmatrix} \begin{bmatrix} \underline{x}_1(L-1) \\ \hline \underline{x}_2(L-1) \end{bmatrix}. \tag{4.28}$$

Die Matrixmultiplikation in (4.28) kann nun folgendermaßen dargestellt werden:

$$\underline{X}_{WH}(L) = \begin{bmatrix} [H_H(L-1)]\ \underline{x}_1(L-1) + [H_H(L-1)]\ \underline{x}_2(L-1) \\ \hline [H_H(L-1)]\ \underline{x}_1(L-1) - [H_H(L-1)]\ \underline{x}_2(L-1) \end{bmatrix}$$

$$= \begin{bmatrix} \underline{X}_{WH1}(L-1) + \underline{X}_{WH2}(L-1) \\ \hline \underline{X}_{WH1}(L-1) - \underline{X}_{WH2}(L-1) \end{bmatrix}. \tag{4.29}$$

Gleichung (4.28) stellt die Beziehung zu den Hadamard-Koeffizienten der Ordnung N/2 her. $\underline{X}_{WH1}$ und $\underline{X}_{WH2}$ beziehen sich auf die obere und die untere Hälfte des Vektors $\underline{X}_{WH}$. Die Substitution (4.29) in (4.27) führt zu

$$\underline{X}_{f<k>}(L) = \begin{bmatrix} [T_{<k>}(L-1)] & 0 \\ \hline 0 & [R_{<k>}(L-1)] \end{bmatrix} \begin{bmatrix} \underline{X}_{WH1}(L-1) + \underline{X}_{WH2}(L-1) \\ \hline \underline{X}_{WH1}(L-1) - \underline{X}_{WH2}(L-1) \end{bmatrix}$$

$$= \begin{bmatrix} [T_{<k>}(L-1)] \cdot [\underline{X}_{WH1}(L-1) + \underline{X}_{WH2}(L-1)] \\ \hline [R_{<k>}(L-1)] \cdot [\underline{X}_{WH1}(L-1) - \underline{X}_{WH2}(L-1)] \end{bmatrix}.$$

Aus (4.26) haben wir

$$\underline{X}_{1f<k>}(L-1) = [T_{<k>}(L-1)]\ \underline{X}_{WH1}(L-1)$$

und $\underline{X}_{2f<k>}(L-1) = [T_{<k>}(L-1)]\ \underline{X}_{WH2}(L-1)$.

Wegen der Struktur von $[T_{<k>}(L)]$ kann man auch schreiben

$$\underline{X}_{WH1}(L-1) = [T_{<k>}(L-1)]\,\underline{X}_{f<k>}(L-1) + [T_{<k>}(L-1)]\,\underline{X}_{1f<k>}(L-1). \qquad (4.30)$$

Aus (4.30) ergibt sich dann

$$\underline{X}_{f<k>} = \begin{bmatrix} \underline{X}_{1f<k>}(L-1) + \underline{X}_{2f<k>}(L-1) \\ \hline [R_{<k>}(L-1)][T_{<k>}(L-1)](\underline{X}_{1f<k>}(L-1) - \underline{X}_{2f<k>}(L-1)) \end{bmatrix}.$$

Zur Vereinfachung ersetzen wir $[R_{<k>}(L-1)][T_{<k>}(L-1)]$ durch $[Z_{<k>}(L-1)]$:

$$\underline{X}_{f<k>}(L) = \begin{bmatrix} \underline{X}_{1f<k>}(L-1) + \underline{X}_{2f<k>}(L-1) \\ \hline [Z_{<k>}(L-1)](\underline{X}_{1f<k>}(L-1) - \underline{X}_{2f<k>}(L-1)) \end{bmatrix}.$$

Nun betrachten wir verschiedene Beispiele von $[T(L)]$ mit $L = 1,2,\ldots,r$ und $N=2^r$:

(1) L=1, N=2:

$$[E_2] = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \qquad [W_2^E] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

$$[T_k(1)] = N^{-1}\,[W_2^E][H_H(1)] = (1/2)\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Durch Bitumkehr folgt:

$$[T_k(1)] = \begin{bmatrix} T_{<k>}(0) & 0 \\ 0 & R_{<k>}(0) \end{bmatrix} = \begin{bmatrix} \underline{T}_0(1) \\ \underline{T}_1(1) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$[T_{<k>}(0)] = [1], \qquad [R_{<k>}(0)] = [1].$$

(2) L=2, N=4:

$$[E_4] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \\ & & & \\ 0 & 2 & 0 & 2 \\ 0 & 3 & 2 & 1 \end{bmatrix}. \qquad [W_4^E] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ & & & \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}.$$

$$[T_k(2)] = \begin{bmatrix} \underline{T}_0(2) \\ \underline{T}_1(2) \\ \\ \underline{T}_2(2) \\ \underline{T}_3(2) \end{bmatrix} = N^{-1} [W_4^E][H_H(2)] = (1/4) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ & & & \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ & & & \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & (1-j)/2 & (1+j)/2 \\ & & & \\ 0 & 1 & 0 & 0 \\ 0 & 0 & (1+j)/2 & (1-j)/2 \end{bmatrix}.$$

Bitumkehr ergibt:

$$[T_{<k>}(2)] = \begin{bmatrix} \underline{T}_0(2) \\ \underline{T}_2(2) \\ \\ \underline{T}_1(2) \\ \underline{T}_3(2) \end{bmatrix}$$

$$= \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & (1-j)/2 & (1+j)/2 \\ 0 & 0 & (1+j)/2 & (1-j)/2 \end{array}\right] = \left[\begin{array}{c|c} T_{<k>}(1) & 0 \\ \hline 0 & R_{<k>}(1) \end{array}\right].$$

$$\text{d.h. } [T_{<k>}(1)] = \begin{bmatrix} 1 & 0 \\ & \\ 0 & 1 \end{bmatrix}, \qquad [R_{<k>}(1)] = \begin{bmatrix} (1-j)/2 & (1+j)/2 \\ & \\ (1+j)/2 & (1-j)/2 \end{bmatrix}.$$

(3) L=3, N=8:

$$[E] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 2 & 4 & 6 & 0 & 2 & 4 & 6 \\ 0 & 3 & 6 & 1 & 4 & 7 & 2 & 5 \\ 0 & 4 & 0 & 4 & 0 & 4 & 0 & 4 \\ 0 & 5 & 2 & 7 & 4 & 1 & 6 & 3 \\ 0 & 6 & 4 & 2 & 0 & 6 & 4 & 2 \\ 0 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{bmatrix}.$$

$$[W_8^E] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1-j & -j & 1+j & -1 & -1+j & j & -1-j \\ 1 & -j & -1 & j & 1 & -j & -1 & j \\ 1 & 1+j & j & 1-j & -1 & -1-j & -j & -1+j \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1+j & -j & -1-j & -1 & 1-j & j & 1+j \\ 1 & j & -1 & -j & 1 & j & -1 & -j \\ 1 & -1-j & j & -1+j & -1 & 1+j & -j & 1-j \end{bmatrix}.$$

$$[H_H(3)] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}.$$

$$[T_k(3)] = N^{-1}[W_8^E][H_H(3)] = \begin{bmatrix} \underline{T}_0(3) \\ \underline{T}_1(3) \\ \underline{T}_2(3) \\ \underline{T}_3(3) \\ \\ \underline{T}_4(3) \\ \underline{T}_5(3) \\ \underline{T}_6(3) \\ \underline{T}_7(3) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & (3-j)/4 & 0 & 0 & (2+3j)/4 \\ 0 & 0 & (1-j)/2 & (1+j)/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & (3+j)/4 & (j-1)/4 & (1+j)/4 & (1-3j)/4 \\ & & & & & & & \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & (-1-j)/4 & (3-j)/4 & (2+3j)/4 & (1-j)/4 \\ 0 & 0 & (1+j)/2 & (1-j)/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & (3+j)/4 & (1-3j)/4 & 0 \end{bmatrix}.$$

Durch Bitumkehr erhält man:

$$[T_{<k>}(3)] = \begin{bmatrix} \underline{T}_0(3) \\ \underline{T}_4(3) \\ \underline{T}_2(3) \\ \underline{T}_6(3) \\ \\ \underline{T}_1(3) \\ \underline{T}_5(3) \\ \underline{T}_3(3) \\ \underline{T}_7(3) \end{bmatrix} = \left[\begin{array}{c|c} [T_{<k>}(2)] & 0 \\ \hline 0 & [R_{<k>}(2)] \end{array}\right] =$$

$$
= \left[\begin{array}{cccc|cccc}
1 & 0 & 0 & 0 & & & & \\
0 & 1 & 0 & 0 & & & & \\
0 & 0 & (1-j)/2 & (1+j)/2 & & 0 & & \\
0 & 0 & (1+j)/2 & (1-j)/2 & & & & \\
\hline
 & & & & (3-j)/4 & 0 & 0 & (2+3j)/4 \\
 & & 0 & & (-1-j)/4 & (3-j)/4 & (2+3j)/4 & (1-j)/4 \\
 & & & & (3+j)/4 & (j-1)/4 & (1+j)/4 & (1-3j)/4 \\
 & & & & 0 & (3+j)/4 & (1-3j)/4 & 0
\end{array}\right].
$$

Mit diesen Beispielen ist die rekursive Erzeugung der Konversionsmatrizen [T(L)] illustriert. Mit $[W_N^E] = [T(L)][H_H(L)]$ folgt also die Möglichkeit der rekursiven Berechnung der Fourier-Koeffizienten der Ordnung L (L = 1,2,...,r und $r=\log_2 N$) durch Koeffizienten der Ordnung L-1, d.h. durch Koeffizienten von zwei Teilfolgen der Längen N/2.

4.2.7 FFT-Algorithmen für reelle symmetrische oder antisymmetrische Datensequenzen

Die DFT transformiert komplexwertige Daten in ebenfalls komplexe DFT-Koeffizienten. Falls die Datenfolge reell ist oder gewisse Symmetrien aufweist, kann man dies zur Steigerung der Effektivität ausnutzen.

Sind die Imaginärteile einer Datenfolge identisch Null, würden bei einem Algorithmus für komplexe Daten etwa 50% der Operation nutzlos ausgeführt, indem Nullen addiert oder mit Null multipliziert würde. Zur effektiven Berechnung der DFT reeller Daten mit einem Algorithmus für komplexe Daten der Länge N hat man zwei Möglichkeiten: Entweder man nutzt den Imaginärteil für die Transformation einer zweiten reellen Funktion der Länge N, oder man transformiert eine reelle Datenfolge der Länge 2N. Dazu teilt man die Datensequenz in Teilsequenzen mit geraden und ungeraden Indizes auf und betrachtet die Folge mit geraden Indizes als Realteil und die mit ungeraden als Imaginärteil.

Im ersten Fall berechnen wir die DFT von

$$x(n) = x_1(n) + jx_2(n),$$

wobei $x_1(n)$ und $x_2(n)$ zwei reelle Folgen der Länge N sind. Die Transformation der komplexen Funktion x(n) ergibt das Spektrum $X_f(k)$ Gemäß Abschnitt 2.1 gilt dann für die Spektren von $x_1(n)$ und $x_2(n)$

$$X_{f1}(k) = X_{RE}(k) + jX_{IO}(k)$$

$$= (1/2)\{X_R(k) + X_R(N-k)\} + j\{X_I(k) - X_I(N-k)\}$$

und

$$X_{f2}(k) = X_{IE}(k) + jX_{RO}(k)$$

$$= (1/2)\{X_I(k) + X_I(N-k)\} - j\{X_R(k) - X_R(N-k)\}.$$

Hierin bezeichnen die Indizes R und I den Real- bzw. den Imaginärteil, sowie E und O die gerade (even) bzw. ungerade (odd) indizierten Koeffizienten. Durch eine einfache Nachverarbeitung (Additionen) der Komplexität O(N) erhält man also die komplexen Spektren $X_{f1}(k)$ und $X_{f2}(k)$.

Falls man andererseits eine reelle Sequenz der Länge 2N anstelle einer komplexen Sequenz der Länge N transformieren will, sind die Daten in eine gerade indizierte Folge und eine ungerade indizierte Folge der Längen N aufzuteilen, die dann als Real- bzw. Imaginärteil der zu transformierenden Sequenz x(n) angegeben werden:

$$x(2n) = x_E(n), \qquad x(2n+1) = x_O(n), \qquad n = 0,1,\ldots,N-1.$$

Aus der transformierten Sequenz der Länge N lassen sich dann DFT-Koeffizienten der ursprünglichen Sequenz der Länge 2N bestimmen [2.1]. Da die Koeffizienten $X_f(k)$ komplex sind, können sie nur bis k = N-1 bestimmt werden:

$$X_{R/2N}(k) = (1/2)\{X_R(k) + X_R(N-k)\} + (1/2)\cos(\pi k/N)\{X_I(k) + X_I(N-k)\}$$

$$- (1/2)\sin(\pi k/N)\{X_R(k) - X_R(N-k)\},$$

$$X_{I/2N}(k) = (1/2)\{X_I(k) - X_I(N-k)\} - (1/2)\sin(\pi k/N)\{X_I(k) + X_I(N-k)\}$$
$$- (1/2)\cos(\pi k/N)\{X_R(k) - X_R(N-k)\},$$
$$k = 0,1,\ldots,N-1.$$

Für beliebig strukturierte reelle 2N-Folgen kann man also nur die ersten N Koeffizienten (nach Betrag und Phase) bestimmen. Die hierzu erforderliche Nachverarbeitung benötigt außer Additionen auch Multiplikationen und die Berechnung (bzw. Abspeicherung) von Sinus- und Cosinus-Werten.

Weitere Vereinfachungen der Berechnung ergeben sich dann, wenn man weitere Annahmen über die Eigenschaften der zu transformierenden Datensequenzen machen kann, z.B. bezüglich Symmetrie oder Antisymmetrie der Datensequenz (vgl. Abschnitt 2.1). Wir betrachten nachfolgend einen Algorithmus, der sich besonders für solche Fälle eignet.

Ein zusätzlicher Flußgraph für die Bestimmung der DFT einer reellen Folge von N Punkten mit einem (komplexen) (N/2)-Punkte-Algorithmus ist im Bild 4.17 angegeben. Dazu werden die gerade indizierten Werte in den Realteil, die ungerade indizierten Werte in den Imagnärteil eines Vektors $x_{N/2}(n)$ eingeordnet. Durch Transformation erhält man die (komplexen) Koeffizienten $X_{N/2}(k)$. Mit Hilfe des im Bild 4.17(a) dargestellten SFG ergeben sich die DFT-Koeffizienten mit $0<k<N/4$. Da sich für jedes k auch ein Koeffizient der Ordnung (N/2)-k ergibt, liefert die Prozedur die erste Hälfte der Koeffizienten. Die zweite Hälfte folgt aus der Kenntnis, daß $X_f(k)$ hermitisch ist. Umgekehrt kann man mit dem (N/2)-Punkte-Algorithmus aus den Werten von $X_{N/2}(k)$ mit Hilfe des SFG vom Bild 4.17(b) und unter Voraussetzung eines hermitisch symmetrischen Spektrums die inverse Transformierte bestimmen.

Bei der Bewertung von Berechnungskosten eines numerischen Algorithmus' wird die Anzahl der arithmetischen Operationen häufig zur Kennzeichnung der Komplexität des Algorithmus' benutzt. Weil Multiplikationen im allgemeinen kostspieligere Operationen sind, ist es oft vorteilhaft, beim Entwurf des Algorithmus mit einer minimalen Anzahl von Multiplikationen auszukommen, ggf. auch auf Kosten der gesamten Anzahl der arithmetischen Operationen. Ein Algorithmus, der für reelle, symmetrische oder antisymmetrische Datensequenzen optimiert ist, wurde von Preuss [4.18] angegeben.

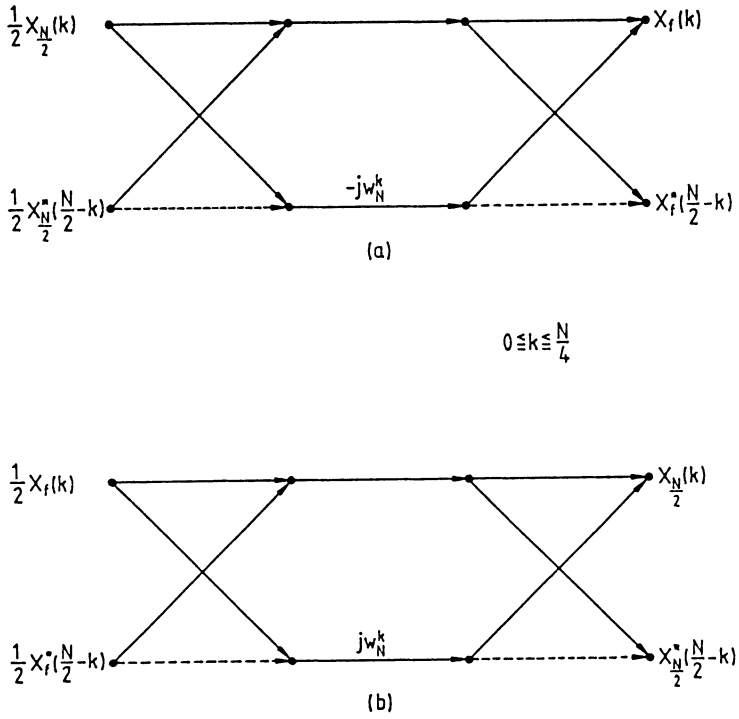


Bild 4.17. Zusätzlicher SFG für die N-DFT mit einem (N/2)-Algorithmus: a) Vorwärtstransformation, b) Rücktransformation, nach [4.28]

Wir betrachten zunächst eine komplexe Sequenz, $x(n)$, $n = 0,\ldots,N-1$, der Länge $N=2^r$. Diese Sequenz läßt sich als

$$x(n) = \{x_{rs}(n) + x_{ra}(n)\} + j\{x_{is}(n) + x_{ia}(n)\} \tag{4.31}$$

mit

$$x_{rs}(n) = \mathrm{Re}\{x(n) + x(N-n)\}/2, \tag{4.32}$$

$$x_{ra}(n) = \mathrm{Re}\{x(n) - x(N-n)\}/2, \tag{4.33}$$

$$x_{is}(n) = \mathrm{Im}\{x(n) + x(N-n)\}/2 \tag{4.34}$$

und $$x_{ia}(n) = \mathrm{Im}\{x(n) - x(N-n)\}/2 \tag{4.35}$$

schreiben. Dabei ist $\mathrm{Re}\{x\} = (x+x^*)/2$, $\mathrm{Im}\{x\} = (x-x^*)/2j$. j ist die imagi-

näre Einheit und das hochgestellte Sternchen symbolisiert komplexe Konjugation. Außerdem wird x(N)=x(0) vorausgesetzt.

Sei $[F_N]$ die DFT-Matrix für N Punkte, so können wir schreiben (vgl. Abschnitt 2.1) $\underline{X}_f = [F_N]\cdot\underline{x} = (X_{RS} + X_{RA}) + j(X_{IS} + X_{IA})$, wobei

$$\underline{X}_{RS} = [F_N]\cdot\underline{x}_{rs}, \qquad \underline{X}_{IS} = [F_N]\cdot\underline{x}_{is},$$
$$\underline{X}_{RA} = [F_N]\cdot\underline{x}_{ra}, \qquad \underline{X}_{IA} = [F_N]\cdot\underline{x}_{ia}. \tag{4.36}$$

Man bemerkt, daß x_{rs} und x_{is} reelle symmetrische Sequenzen sind, für die folgende Beziehung gilt

$$x(n) = x(N-n) = x^*(n), \qquad n = 0,1,\ldots,(N/2).$$

Außerdem beachte man, daß x_{ra} und x_{ia} reelle antisymmetrische Sequenzen sind, d.h. für sie gilt

$$x(n) = -x(N-n) = x^*(n), \qquad n = 0,1,\ldots,(N/2).$$

Es ist leicht zu zeigen, daß die DFT einer reellen symmetrischen Sequenz eine reelle symmetrische Sequenz ist. Darüber hinaus ist die DFT einer reellen antisymmetrischen Sequenz eine rein imaginäre antisymmetrische Sequenz:

$$\begin{aligned} \mathrm{Re}\{X_f(k)\} &= X_{RS}(k) + jX_{IA}(k), \\ \mathrm{Im}\{X_f(k)\} &= X_{IS}(k) - jX_{RA}(k). \end{aligned} \tag{4.37}$$

Der folgende Algorithmus läßt sich zur Berechnung der Transformationen von (4.36) verwenden, die dann in (4.37) zur Berechnung der Transformation der ursprünglichen Sequenz benutzt werden können.

Es wurde bereits erwähnt, daß wenn x(n) reell und (anti-)symmetrisch ist, die Berechnung der N-Punkte-DFT auf die Berechnung einer (N/2)-Punkte-DFT der reellen Datensequenz $y(n) = a_0(n) + a_1(n)$ zurückgeführt werden kann, wobei $n = 0,1,\ldots,(N/2)-1$, und

$$a_0(n) = x(2n), \quad a_1(n) = x(2n+1) - \beta\cdot x(2n-1) \quad \text{sowie} \quad x(-1) = x(N-1)$$

gilt. Die Größe ß kann die Werte +1 oder −1 annehmen. Zunächst sei ß = +1. Für eine symmetrische bzw. antisymmetrische Sequenz x(n) gilt

$$x(n) = S \cdot x(N-n) = x^*(n) \quad \text{mit} \quad S = \pm 1. \tag{4.38}$$

Wenn $Y_f(k)$ die Transformierte von y(n) ist, erhält man $A_0(k)$ und $A_1(k)$ (die Transformierten von $a_0(0)$ und $a_1(n)$) aus

$$A_{1-S/2}(k) = \mathrm{Re}\{Y_f(k)\} \quad \text{bzw.} \quad A_{1+S/2}(k) = j\,\mathrm{Im}\{Y_f(k)\}. \tag{4.39}$$

Die Berechnung von $X_f(k)$, d.h. der Transformierten von x(n), aus $A_0(k)$ und $A_1(k)$ gestaltet sich dann wie folgt [4.18]:

$$R = (1+\beta S) \sum_{n=0}^{(N/4)-1} x(2n+1)\beta^{n+1} = \sum_{n=0}^{(N/2)-1} x(2n+1)\beta^{n+1} \tag{4.40}$$

sowie $B(0) = \{(1+\beta)2R + (1-\beta)A_1(0)\}/4$

und $B(N/4) = j\{(1-\beta)2R - (1+\beta)A_1(N/4)\}/4.$

Dann sei $B(k) = (W_N^{-k} - \beta \cdot W_N^k) - A_1(k)$ für $k = 1,2,\ldots,(N/4)-1.$ (4.41)

Damit ist $X_f(k) = A_0(k) + B(k)$ für $k = 0,1,\ldots,(N/4)$ (4.42)

und $X_f((N/2)-k)) = S \cdot (A_0(k) - B(k))$ für $k = 1,2,\ldots,(N/4)-1$ (4.43)

mit $X_f(N/2) = A_0(0) - B(0).$ (4.44)

Für $k = 1,2,\ldots,(N/2)-1$ folgt schließlich

$$X_f(N-k) = S \cdot X_f(k) = X^*(k). \tag{4.45}$$

Man beachte, daß die einzigen nicht-trivialen Multiplikationen im obigen Verfahren, außer den zur Erzeugung von $A_0(k)$ und $A_1(k)$ erforderlichen, die für (4.41) benötigten (N/4)−1 reellen Multiplikationen sind.

Von (4.39) ausgehend sehen wir, daß $A_0(k)$ und $A_1(k)$ einfach aus $Y_f(k)$ erzeugt werden können. Wenn ß=+1, können sie aus einer einzigen (N/2)-DFT der

reellen Sequenz y(n) berechnet werden. $a_0(n)$ und $a_1(n)$ sind jedoch symmetrische oder antisymmetrische Sequenzen der Länge N/2, so daß ihre Transformierten unter Verwendung desselben Prozesses wie zur Berechnung der DFT von x(n) zu ermitteln sind.

Bisher wurde der Fall ß=+1 und S=±1 untersucht. Es gibt jedoch noch den Fall ß=−1, S=±1, d.h. wir haben vier Situationen: Entweder positive (ß=+1) Zerlegung einer symmetrischen (S=+1) oder antisymmetrischen (S=−1) Sequenz oder die entsprechenden negativen Zerlegungen (ß=−1).

Preuss [4.18] diskutiert die positive Zerlegung für symmetrische Sequenzen und die negative Zerlegung für antisymmetrische Sequenzen. Für eine die Gleichung (4.38) erfüllende Sequenz wählt er also ß = S. Dann wird

$$a_0((N/2)-n) = x(N-2n) = S\cdot x(2n) = S\cdot a_0(n) \qquad (4.46)$$

und unter Verwendung des Algorithmus' (4.40) bis (4.45) kann ihre Transformierte durch (N/8)−1 Multiplikationen ermittelt werden, indem man

$$a_{00}(n) = a_0(2n) \quad \text{und} \quad a_{01}(n) = a_0(2n+1) - S\cdot a_0(2n-1) \qquad (4.47)$$

für n = 0,1,...,(N/4)−1 berechnet.

Analog dazu erfüllt $a_1(n)$ die Beziehungen

$$a_1((N/2)-n) = x(N-2n+1) - S\cdot x(N-2n+1) = S\cdot x(2n-1) - x(2n+1) = -a_1(n), \qquad (4.48)$$

so daß ihre Transformierte durch

$$a_{10}(n) = a_1(2n) \quad \text{und} \quad a_{11}(n) = a_1(2n+1) + a_1(2n-1) \qquad (4.49)$$

für n = 0,1,...,(N/4)−1 berechnet werden kann.

Die DFT der N-Punkte-Sequenz x(n) kann also unter Verwendung der DFT der vier durch (4.47) und (4.48) gegebenen (N/4)-Punkte-Sequenzen und den zusätzlich benötigten {(N/4)−1+2((N/8)−1)} Multiplikationen und {N−3+S+2((N/2)−4+(1+S)/2)} Additionen berechnet werden. Die durch (4.46),

(4.47) und (4.49) gegebenen Sequenzen sind voraussetzungsgemäß reelle symmetrische oder antisymmetrische Sequenzen. Man kann also die DFT einer reellen symmetrischen oder antisymmetrischen N-Punkte-Sequenz x(n) aus den Transformierten von 2^p reellen symmetrischen oder antisymmetrischen Sequenzen der Längen $N \cdot 2^{-p}$ berechnen. Zusätzlich benötigt man

$$\sum_{m=0}^{p} 2^{\beta m-1} \cdot (2^{1-m}(N/4)-1) = p \cdot (N/4) - 2^p + 1 \qquad (4.50)$$

Multiplikationen und

$$\sum_{m=0}^{p} 2^{\beta m-1} (\ (2^{1-m} \cdot N - 4) + (1+S) \ ) = p \cdot (N + 1 + S) - 4(2^p-1)$$

Additionen.

Bisher wurde gezeigt, daß zur Berechnung der DFT einer reellen symmetrischen oder antisymmetrischen Sequenz folgende Operationen benötigt werden: 2^p-mal die Anzahl der Multiplikationen und Additionen für die Transformation (4.50) einer reellen symmetrischen oder antisymmetrischen $N \cdot 2^{-p}$-Punkte-Sequenz plus $p(N/4)-2^p+1$ Multiplikationen, plus $p(N+1+S)-4 \cdot (2^p-1)$ Additionen. Die Transformation (4.50) einer komplexen N-Punkte-Sequenz erfordert im allgemeinen viermal soviele Operationen wie die obige Anzahl von Operationen plus 4N-4 Additionen für Vor- und Nachverarbeitung der komplexen Sequenz. Dies bringt die Gesamtzahl auf

$$M_c(N,p) = 4\{p(N/4)-2^p+1\} + (2^{p+2}-2)M_a(N2^{-p},0) + 2M_s(N2^{-p},0) \qquad (4.51)$$

reelle Multiplikationen plus

$$A_c(N,p) = 4\{p(N+1)-4(2p-1)\} + 4N - 4 + (2^{p+2})A_a(N2^{-p},0) + 2A_s(N2^{-p},0) \qquad (4.52)$$

reelle Additionen für die DFT einer komplexen N-Punkte-Sequenz. Dabei bedeuten M_c und A_c die Anzahl der Multiplikatonen bzw. Additionen einer komplexen Transformation, während die Indizes s und a symmetrische bzw. antisymmetrische Transformationen bezeichnen. M_a und M_s bzw. A_a und A_s stellen die Anzahl der benötigten Multiplikationen und Additionen zur Berechnung der N-Punkte-DFT von rellen symmetrischen bzw. reellen antisymmetrischen Datensequenzen dar.

Normalerweise wird p so gewählt, daß die Kostenfunktion

$$C(N,p) = C_M M_c(N,p) + C_A A_c(N,p)$$

ein Minimum wird, wobei C_M die Kosten einer reellen Multiplikation und C_A die Kosten einer Addition oder Substraktion sind.

Im Folgenden ist der komplette Algorithmus zu diskutieren, der unter Annahme ($C_M >> C_A$) erzeugt wird, so daß p wegen

$$M_a(4,0) = M_s(4,0) = 0$$

als $p = \log_2(N/4)$

gewählt wird. Für diesen Fall ergeben (4.54) und (4.51)

$$M_c(N,\log_2(N/4)) = N\cdot\log_2(N/4) - N + 4$$

reelle Multiplikationen und

$$A_c(N,\log_2(N/4)) = 4(N+1)\cdot\log_2(N/4) + N + 20$$

reelle Additionen, wobei $A_S(4,0) = 5$ und $A_A(4,0) = 1$ benutzt wurden.

Nun führen wir den Algorithmus konkrekt aus. Der erste Schritt im Algorithmus ist, die komplexe Sequenz (4.31) auf die vier Sequenzen (4.32), (4.33), (4.34) und (4.35) zu reduzieren. Wegen der Voraussetzung der Symmetrie sind nur 2N–4 reelle Additionen zur Erzeugung der vier Sequenzen erforderlich. Außerdem erhalten wir:

$$x_{ra}(0) = x_{ia}(0) = x_{ra}(N/2) = x_{ia}(N/2) = 0,$$

$$x_{rs}(0) = \mathrm{Re}\{x(0)\}, \qquad x_{rs}(N/2) = \mathrm{Re}\{x(N/2)\},$$

$$x_{is}(0) = \mathrm{Im}\{x(0)\}, \qquad x_{is}(N/2) = \mathrm{Im}\{x(N/2)\}.$$

Während (N/2)+1 Speicherplätze zur Darstellung der Sequenzen $x_{rs}(n)$ und $x_{is}(n)$ erforderlich sind, werden nur (N/2)–1 Speicherplätze zur Darstellung

von $x_{rs}(n)$ und $x_{is}(n)$ benötigt. Deshalb kann dieser Teil des Algorithmus' vollständig mit Ersetzung ("in-place") ausgeführt werden. Im zweiten Schritt des Algorithmus' wird nun jede der vier Sequenzen der Reihe nach transformiert, wobei der transformierte Vektor auf den gleichen Platz gebracht wird, den zuvor der Eingabevektor belegt hatte.

Der dritte Schritt des Algorithmus' erzeugt die Real- und Imaginärteile gemäß Gleichung (4.37). $X_{RS}(k)$ und $X_{IS}(k)$ sind rein reell und symmetrisch, während $X_{RA}(k)$ und $X_{IA}(k)$ rein imaginär und antisymmetrisch sind. Eine Betrachtung von (4.37) zeigt, daß dieser Schritt "in-place" mit nur 2N Additionen durchgeführt werden kann. Der wesentliche Teil der Transformation erfolgt im zweiten Schritt. Dabei ist die Eingabe ein Vektor x(n) der Länge (N/2)+S. Die Ausgabe ist ein Vektor der Länge (N/2)+S, der die Transformierte der Eingabesequenz repräsentiert. Wenn die Eingabesequenz antisymmetrisch (S=-1) ist und die positive Zerlegung (ß=+1) benutzt wird, werden eine symmetrische und eine antisymmetrische Sequenz erzeugt, die

$$((N/4)+1) + ((N/4)-1) = N/2$$

Speicherplätze (d.h. einen mehr als die Anzahl der für die Eingabesequenz benötigten) erfordert. Bei negativer Zerlegung (ß=-1) werden zwei antisymmetrische Sequenzen erzeugt (erforderliche Speicherplätze (N/2)-2) sowie ein zusätzlicher zu speichernder (durch (4.40) gegebener) Wert. Diese Situation entsteht für ß=S=-1.

Bei symmetrischer Eingabesequenz (S=+1) ist die Situation umgekehrt. Die Anwendung der negativen Zerlegung (ß=-1) produziert zwei symmetrische Sequenzen, die einen Bedarf an einem zusätzlichen Speicherplatz erzeugt, während die positive Zerlegung (ß=+1) eine symmetrische und eine antisymmetrische Sequenz plus einem zusätzlichen durch (4.40) gegebenen Wert produziert. Für ß=S=+1 tritt also kein zusätzlicher Speicherbedarf auf.

Wenn man also bei der iterativen Zerlegung von symmetrischen (S=+1) bzw. antisymmetrischen (S=-1) Sequenzen positive (ß=+1) bzw. negative (ß=-1) Zerlegung anwendet, entsteht kein zusätzlicher Speicherbedarf. Bei vollständiger Zerlegung produziert eine komplexe N-Punkte-Sequenzen zwei reelle symmetrische 4-Punkte-Sequenzen und N-2 reelle antisymmetrische 4-Punkte-Sequenzen.

Die Erstellung der Transformierten erfolgt durch den in (4.40) bis (4.44) dargestellten Prozeß. Ein FORTRAN-Programm, das diesen Algorithmus implementiert, ist in [4.18] enthalten.

Die Gesamtzahl von reellen Multiplikationen zur Berechnung der DFT einer beliebigen komplexen Sequenz der Länge $N=2^r$ ist $M_c(N) = N \cdot \log_2(N/4) - N + 4$ Wenn die Sequenz rein reell ist, sind nur halb so viele Multiplikationen erforderlich. Falls die reelle Sequenz auch symmetrisch oder antisymmetrisch ist, werden nur 1/4 der Multiplikationen benötigt.

Eine weitere Möglichkeit, den Mehraufwand für nicht vorhandene Imaginärteile zu umgehen, bietet die Hartley-Transformation (vgl. Abschnitt 4.3.7). Die Cosinus- und die Sinus-Transformation (vgl. die Abschnitte 2.3, 4.4 und 4.5) können ebenfalls als Spezialfälle der Fourier-Transformation reeller Daten betrachtet und in ähnlicher Weise programmiert werden [4.19].

4.3 Algorithmen für die diskrete Hartley-Transformation

Die Vorteile der diskreten Hartley-Transformation (DHYT) gegenüber der diskreten Fourier-Transformation für den Fall reeller Eingangswerte wurden bereits im Abschnitt 2.2 erläutert. Allgemein hat die Hartley-Transformation zwei Vorteile gegenüber der FFT: (1) Vorwärts- und Rückwärtstransformation sind gleich und (2) Transformationskern und -koeffizienten sind nicht komplex sondern reellwertig. Deshalb kann es zweckmäßig sein, das Fourier-Spektrum durch die Hartley-Transformation zu berechnen (vgl. Abschnitt 4.3.7). In den folgenden Abschnitten geben wir Ableitungen verschiedener Algorithmen zur Berechnung der DHYT an. Es sind dies die verallgemeinerten Versionen des Ansatzes zur Zeitdezimierung sowie zur Frequenzdezimierung, der Primfaktor-Algorithmus [4.20], der Split-Radix-Algorithmus [4.21], sowie die Berechnung aus anderen Transformationskoeffizienten. Alle diese Algorithmen lassen eine "in-place"-Berechnung zu. Dieser Umstand gestattet eine relativ einfache Entwicklung und Optimierung der DHYT-Module. Andererseits muß festgestellt werden, daß die Verwendung der DHYT zur Berechnung der DFT keine Vorteile hinsichtlich der Anzahl der Operationen bietet [4.23].

Gemäß Abschnitt 2.2 gilt für die Beziehung zwischen den DFT-Koeffizienten $X_f(k)$ und den DHYT-Koeffizienten $X_{HY}(k)$ der Datensequenz $x(n)$:

$$X_{HY}(k) = \sum_{n=0}^{N-1} x(n)\,\mathrm{cas}(2\pi kn/N)$$

$$= \sum_{n=0}^{N-1} x(n)\cos(2\pi kn/N) + \sum_{n=0}^{N-1} x(n)\sin(2\pi kn/N)$$

$$= \mathrm{Re}\{X_f(k)\} - \mathrm{Im}\{X_f(k)\}, \qquad k = 0,1,\ldots,N-1. \tag{4.53}$$

Wir weisen noch darauf hin, daß die schnelle Hartley-Transformation in den USA patentiert ist.

4.3.1 DHYT-Algorithmus mit Zeitdezimierung

Zunächst betrachten wir eine Herleitung eines verallgemeinerten Algorithmus' mit Zeitdezimierung [4.20], [4.22]. Um alle Spezialfälle zu erfassen, soll sich die Länge N der Datenfolge als Produkt von Primfaktoren darstellen lassen.

Sei

$$N = P_1 P_2 P_3 \ldots P_r. \tag{4.54}$$

Ohne Verlust der allgemeinen Gültigkeit wird angenommen

$$N = N_1 \cdot N_2 \quad \text{mit} \quad N_1 = P_1 \quad \text{und} \quad N_2 = P_2 P_3 P_4 \ldots P_r. \tag{4.55}$$

Nun wird (4.53) zerlegt:

$$X_{HY}(k) = \sum_{n_1=0}^{N_2-1} x(n_1 N_1)\,\mathrm{cas}(2\pi k n_1 N_1/N)$$

$$+ \sum_{n_1=0}^{N_2-1} x(n_1 N_1+1)\,\mathrm{cas}(2\pi k(n_1 N_1+1)/N) +$$

$$+ \ \ldots \ + \sum_{n_1=0}^{N_2-1} x(n_1N_1+(N_1-1))\ \mathrm{cas}(2\pi k(n_1N_1+(N_1-1))/N). \tag{4.56}$$

Im Folgenden benutzt man die Identität

$$\mathrm{cas}(2\pi(\alpha+\beta)/N) = \cos(2\pi\beta/N)\mathrm{cas}(2\pi\alpha/N) + \sin(2\pi\beta/N)\mathrm{cas}(-2\pi\alpha/N). \tag{4.57}$$

Zunächst wird eine N_2–Punkte–DHYT definiert

$$Y_{N2}(k) = \sum_{n_1=0}^{N_2-1} x(n_1N_1+n_2)\mathrm{cas}(2\pi kn_1N_1/N). \tag{4.58}$$

Wenn hierin k und n_1 durch $-k$ bzw. $-n_1$ ersetzt werden, erhält man

$$Y_{N2}(N_2-k) = \sum_{n_1=0}^{N_2-1} x(n_2-n_1N_1)\mathrm{cas}(2\pi kn_1N_1/N). \tag{4.59}$$

Unter Verwendung von (4.57), (4.58) und (4.59) kann nun (4.56) wie folgt umgeschrieben werden:

$$X_{HY}(k) = \sum_{n_2=0}^{N_1-1} \{\, Y_{N2}(k)\cos(2\pi kn_2/N) + Y_{N2}(N_2-k)\sin(2\pi kn_2/N)\}. \tag{4.60}$$

In dieser Darstellung ist n in n_1 und n_2 aufgespalten

$$n = N_1n_1 + n_2. \tag{4.61}$$

Zur Durchführung der Methode mit Zeitdezimierung wird nun k in

$$k = k_1 + N_2k_2 \tag{4.62}$$

zerlegt, mit $k_1 = 0,1,\ldots,N_2-1$ und $k_2 = 0,1,\ldots,N_1-1$. Dann folgt aus (4.60), (4.61) und (4.62)

$$X_{HY}(N_2k_2+k_1) = \sum_{n_2=0}^{N_1-1} \{ Y_{N2}(k_1)\cos(2\pi k_1 n_2/N) + Y_{N2}(N_2-k_1)\sin(2\pi k_1 n_2/N) \} \cos(2\pi k_2 n_2/N_1)$$

$$+ \sum_{n_2=0}^{N_1-1} \{ Y_{N2}(N_2-k_1)\cos(2\pi k_1 n_2/N) - Y_{N2}(k_1)\sin(2\pi k_1 n_2/N) \} \sin(2\pi k_2 n_2/N_1). \qquad (4.63)$$

Entsprechend findet man für

$$X_{HY}(N-k) = X_{HY}(N-(N_2k_2+k_1)) = \sum_{n_2=0}^{N_1-1} \{ Y_{N2}(N_2-k_1)\cos(2\pi k_1 n_2/N) - Y_{N2}(k_1)\sin(2\pi k_1 n_2/N) \} \cos(2\pi k_2 n_2/N_1)$$

$$- \sum_{n_2=0}^{N_1-1} \{ Y_{N2}(k_1)\cos(2\pi k_1 n_2/N) + Y_{N2}(N_2-k_1)\sin(2\pi k_1 n_2/N) \} \sin(2\pi k_2 n_2/N_1). \qquad (4.64)$$

Aus (4.63) und (4.64) läßt sich ein Signalflußgraph mit "in-place"-Eigenschaft für die DHYT konstruieren. Ein Beispiel mit $N_1=2$ wird durch Bild 4.18 illustriert. Wenn speziell $N=4$ betrachtet wird, ergeben sich folgende Verhältnisse:

$$N_2 = 2, \quad \cos(2\pi k_1/N) = 0, \quad \sin(2\pi k_1/N) = 1$$

$$\left.\begin{aligned} n &= 2n_1 + n_2 \\ k &= 2k_2 + k_1 \end{aligned}\right\} \quad \text{(Bitumkehr)}.$$

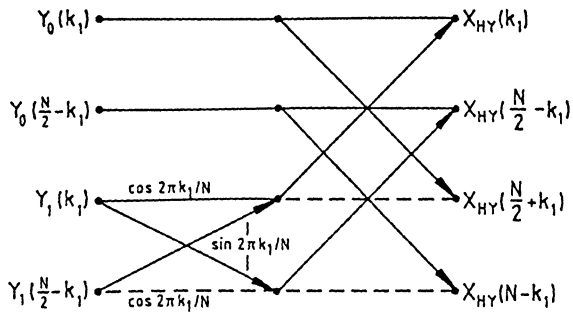


Bild 4.18. Radix-2-Signalflußgraph der DHYT mit Zeitdezimierung, nach [4.20]

4.3.2 DHYT-Algorithmus mit Freqenzdezimierung

Die Ableitung beginnt mit der gleichen Faktorisierung von N wie in (4.54) und (4.55). Nun wird (4.53) wie folgt zerlegt [4.20], [4.22], [4.23], [4.24]:

$$X_{HY}(k) = \sum_{n=0}^{N_2-1} x(n)\,\mathrm{cas}(2\pi kn/N) + \sum_{n=N_2}^{2N_2-1} x(n)\,\mathrm{cas}(2\pi kn/N)$$

$$+ \ldots + \sum_{n=(N_1-1)N_2}^{N_1N_2-1} x(n)\,\mathrm{cas}(2\pi kn/N). \qquad (4.65)$$

Für die i-te Summe wird n durch

$$n = n_1 + (i-1)N_2$$

ersetzt. Damit erhält man

$$X_{HY}(k) = \sum_{n_1=0}^{N_2-1} x(n_1)\,\mathrm{cas}(2\pi kn_1/N)$$

$$+ \ldots + \sum_{n_1=0}^{N_2-1} x(n_1+N_2)\,\mathrm{cas}(2\pi k(n_1+N_2)/N) +$$

$$+ \ldots + \sum_{n_1=0}^{N_2-1} x(n_1+(N_1-1)N_2)\,\mathrm{cas}(2\pi k(n_1+(N_1-1)N_2)/N).$$

Diese Gleichung kann als Doppelsumme geschrieben werden

$$X_{HY}(k) = \sum_{n_2=0}^{N_1-1} \sum_{n_1=0}^{N_2-1} x(n_1+n_2N_2)\,\mathrm{cas}(2\pi k(n_1+n_2N_2)/N). \qquad (4.66)$$

Mit der Zerlegung

$$k = N_1k_1 + k_2, \quad k_1 = 0,1,\ldots,N_2-1 \quad \text{und} \quad k_2 = 0,1,\ldots,N_1-1 \qquad (4.67)$$

erhält man aus (4.65), (4.66) und (4.67)

$$X_{HY}(N_1k_1+k_2) = \sum_{n_1=0}^{N_2-1} \sum_{n_2=0}^{N_1-1} \{ x(n_2N_2+n_1)\cos(2\pi k_2(N_2n_2+n_1)/N)$$

$$+ x(n_2N_2-n_1)\sin(2\pi k_2(N_2n_2-n_1)/N) \}\,\mathrm{cas}(2\pi k_1n_1/N_2). \qquad (4.68)$$

Analog zum Algorithmus mit Zeitdezimierung benötigt man zum Aufbau der "in-place"-Module

$$X_{HY}(N-(N_1k_1+k_2)) = \sum_{n_1=0}^{N_2-1} \sum_{n_2=0}^{N_1-1} \{x(n_2N_2-n_1)\cos(2\pi k_2(N_2n_2-n_1)/N)$$

$$- x(n_2N_2+n_1)\sin(2\pi k_2(N_2n_2+n_1)/N)\}\,\mathrm{cas}(2\pi k_1n_1/N_2). \qquad (4.69)$$

In ähnlicher Weise wie beim Algorithmus mit Zeitdezimierung können nun die Gleichungen (4.68) und (4.69) umgeschrieben werden als

$$X_{HY}(N_1k_1+k_2) = \sum_{n_1=0}^{N_2-1} \mathrm{cas}(k_1n_1) \{ \sum_{n_2=0}^{N_1-1} [x(n_2N_2+n_1)\ \cos(2\pi k_2n_1/N) -$$

$$- x(n_2N_2-n_1)\sin(2\pi k_2 n_1/N)]\ \cos(2\pi k_2 n_2/N_1)$$

$$- [x(n_2N_2-n_1)\cos(2\pi k_2 n_1/N)$$

$$- x(n_2N_2+n_1)\sin(2\pi k_2 n_1/N)]\sin(2\pi k_2 n_2/N_1)\}$$

und

$$X_{HY}(N-(N_1k_1+k_2)) = \sum_{n_1=0}^{N_2-1} \operatorname{cas}(k_1 n_1)\ \{\sum_{n_2=0}^{N_1-1} [x(n_2N_2-n_1)\ \cos(2\pi k_2 n_1/N)$$

$$- x(n_2N_2+n_1)\sin(2\pi k_2 n_1/N)]\cos(2\pi k_2 n_2/N_1)$$

$$- [x(n_2N_2+n_1)\cos(2\pi k_2 n_1)/N)$$

$$- x(n_2N_2-n_1)\sin(2\pi k_2 n_1)/N)]\sin(2\pi k_2 n_2/N_1)\}.$$

Die Anzahl der Operationen ist für beide Algorithmen gleich. Sie beträgt für die Radix-2-Versionen

$$A_r(N) = (3N/2)(\log_2 N-1) + 2 \quad \text{und} \quad M_r(N) = N(\log_2 N-3) + 4.$$

4.3.3 Primfaktor-Algorithmus für die DHYT

Ein anderer wichtiger Algorithmus für die DHYT ist der Primfaktor-Algorithmus (PFA). Für die Primfaktorindexberechnung ist das Chinesische Rest-Theorem (CRT vgl. Abschnitt 3.2) ein wichtiges Werkzeug. Hier betrachten wir den PFA der DHYT unter Verwendung des CRT ähnlich wie bei der DFT.

Dazu schreibt man (4.53) geeignet um [4.20]

$$X_{HY}(\langle k\rangle_N) = \sum_{n=0}^{N-1} x(\langle n\rangle_N)\operatorname{cas}(2\pi\langle kn\rangle_N/N). \tag{4.70}$$

Dabei ist $N = N_1N_2$, wobei N_1 und N_2 teilerfremd sind. Die Bezeichnung $\langle z\rangle_N$ steht dabei für z mod N. Unter Verwendung des CRT zerlegen man n so, daß

$$n_1 = \langle n\rangle_{N1}, \quad n_2 = \langle n\rangle_{N2} \quad \text{und} \quad \langle n\rangle_N = \langle n_1C_1N_2 + n_2C_2N_1\rangle_N$$

gilt (vgl. Abschnitt 4.2.3), wobei $\langle n_1 C_2 \rangle_{N2} = 1$ und $\langle n_2 C_1 \rangle_{N1} = 1$.

Eine entsprechende Zerlegung gilt für k:

$$k_1 = \langle k \rangle_{N1}, \quad k_2 = \langle k \rangle_{N2}, \quad \text{und} \quad \langle k \rangle_N = \langle k_1 C_3 N_2 + k_2 C_4 N_1 \rangle_N,$$

$$\text{sowie } \langle k_2 C_3 \rangle_{N1} = 1 \quad \text{und} \quad \langle k_1 C_4 \rangle_{N2} = 1. \tag{4.71}$$

In (4.70) eingesetzt erhält man

$$X_{HY}(\langle k \rangle_N) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(\langle n \rangle_N) \cas(2\pi \langle \langle k_1 C_3 N_2 \rangle_N \langle n_1 C_1 N_2 \rangle_N \rangle_N / N). \tag{4.72}$$

Hinsichtlich des CRT gelten die folgenden Identitäten

$$\langle a + b \rangle_N = \langle \langle a \rangle_N + \langle b \rangle_N \rangle_N \quad \text{und} \quad \langle a \cdot b \rangle_N = \langle \langle a \rangle_N \cdot \langle b \rangle_N \rangle_N.$$

Im Transformationskern können deshalb alle Terme fortgelassen werden, die Vielfache von N sind. Aus Gleichung (4.72) wird dann

$$X_{HY}(\langle k \rangle_N) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(\langle n \rangle_N \cas(2\pi \langle k_2 n_2 C_2 C_4 N_1^2 + k_1 n_1 C_1 C_3 N_2^2 \rangle_N / N). \tag{4.73}$$

Unter Verwendung von (4.57) und (4.71) wird (4.73) umgeformt in

$$X_{HY}(\langle k \rangle_N) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \{ x(\langle n \rangle_N) \, [\cos(\langle 2\pi k_1 n_1 C_1 / N_1 \rangle_{N1}) \cas(2\pi \langle k_2 n_2 C_2 / N_2 \rangle_{N2} / N_2)$$

$$+ \sin\langle 2\pi k_1 n_1 C_1 / N_1 \rangle_{N1} \cas(-2\pi \langle k_2 n_2 C_2 / N_2 \rangle_{N2} / N_2)] \}.$$

Mit $C_1 = C_2 = 1$ erhält man schließlich die Gleichungen

$$X_{HY}(\langle k_1 C_3 N_2 + k_2 C_4 N_1 \rangle_N / N) =$$

$$\sum_{n_1=0}^{N_1-1} Z_{n1}(k_2)\cos(2\pi k_1 n_1/N_1) + Z_{n1}(N_2-k_2)\sin(2\pi k_1 n_1/N_1)$$

und $X_{HY}(<k_1 C_3 N_2 - k_2 C_4 N_1>_N) =$

$$\sum_{n_1=0}^{N_1-1} Z_{n1}(N_2-k_2)\cos(2\pi k_1 n_1/N_1) + Z_{n1}(k_2)\sin(2\pi k_1 n_1/N_1)$$

$$Z_{n1}(k_2) = \sum_{n_2=0}^{N_2-1} x(<n_1 N_2 + n_2 N_1>_N)\ \mathrm{cas}(2\pi k_2 n_2/N_2),$$

$$Z_{n1}(N_2-k_2) = \sum_{n_2=0}^{N_2-1} x(<n_1 N_2 - n_2 N_1>_N)\ \mathrm{cas}(2\pi k_2 n_2/N_2).$$

Damit ist die erste Iteration vollständig. Ähnlich wie bei den beiden zuvor beschriebenen Algorithmen werden hieraus die PFA-Module konstruiert und gemäß ihrer "in-place"-Eigenschaft weiter verarbeitet.

4.3.4 Split-Radix-Algorithmus für die DHYT

Im Abschnitt 4.2.2 haben wir den Split-Radix-Algorithmus der FFT diskutiert. Nun betrachten wir diesen Algorithmus für die DHYT [4.21]. Der Split-Radix-Algorithmus benötigt die niedrigste Anzahl von Operationen unter allen schnellen Hartley-Transformationsalgorithmen.

Wir erinnern daran, daß der Split-Radix-Algorithms auf Radix-2-Zerlegungen für die geraden Anteile sowie auf Radix-4-Zerlegungen für die ungeraden Anteile beruht. Wenn die diskrete Hartley-Transformation (DHYT)

$$X_{HY}(k) = \sum_{n=0}^{N-1} x(n)\mathrm{cas}(2\pi kn/N)$$

zu berechnen ist, wird diese Gleichung wie folgt zerlegt:

$$X_{HY}(2k) = \sum_{n=0}^{(N/2)-1} \{x(n)+x(n+(N/2))\}\mathrm{cas}(2\pi kn/N),$$

$$X_{HY}(4k+1) = \sum_{n=0}^{(N/4)-1} \{[A(n)+A((N/4)-n)]\cos(2\pi n/N) - [B(n)-B((N/4)-n)\sin(2\pi n/N)]\,cas(2\pi kn/(N/4))\},$$

$$X_{HY}(4k+3) = \sum_{n=0}^{(N/4)-1} \{[A(n)-A((N/4)-n)]\cos(2\pi 3n/N) - [B(n)+B((N/4)-n)\sin(2\pi 3n/N)]\,cas(2\pi kn/(N/4))\},$$

wobei $A(n) = x(n) - x(n+(N/2))$, $A(N/4) = B(0)$ und

$$B(n) = x(n+(N/4)) - x(n+(3N/4)).$$

Die o.g. Zerlegung mit Frequenzdezimierung reduziert eine N-Punkte-DHYT zu einer (N/2)-Punkte-DHYT auf Kosten von (N-1) reellen Multiplikationen und (2N-4) reellen Additionen, d.h.

N-Punkte-DHYT ——> (N/2)-Punkte-DHYT + 2x(N/4)-Punkte-DHYT .

Bild 4.19 zeigt einen Signalflußgraph für die Zerlegung einer 16-Punkte-DHYT in eine 8-Punkte-DHYT und zwei 4-Punkte-DHYT auf Kosten von 10 reellen Multiplikationen und 26 reellen Additionen. Eine N-Punkte-DHYT läßt sich dann durch iterative Wiederholung solcher Zerlegungen bis zur letzten Iteration erzeugen.

Bild 4.20 stellt eine vollständige 8-Punkte-DHYT dar. Die Koeffizienten sind wie bei allen anderen schnellen DHYT-Algorithmen mit Frequenzdezimierung in Bitumkehrordnung.

Die Split-Radix-Hartley-Transformation besitzt eine geringere Berechnungskomplexität als die in den Abschnitten 4.3.1 und 4.3.2 beschriebenen DHYT-Algorithmen. Die Gesamtzahl der Operationen der Split-Radix-DHYT sind

$$A_r(N) = (2N-4)\lceil \log_4 N \rceil \quad \text{Additionen}$$

und

$$M_r(N) = (N-4)\lceil \log_4 N \rceil \quad \text{Multiplikationen.}$$

Der Algorithmus ist "in-place" realisierbar.

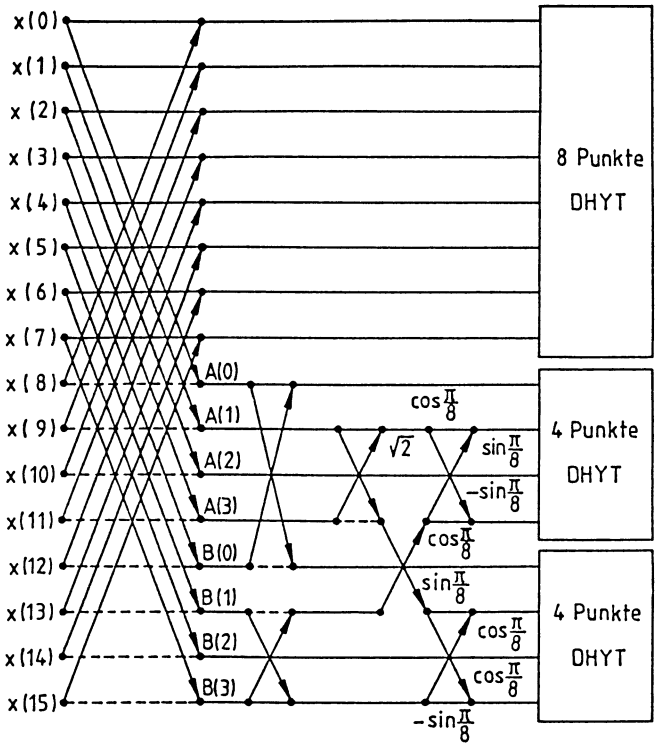


Bild 4.19. Split-Radix-Signalflußgraph der DHYT für N=16, nach [4.21]

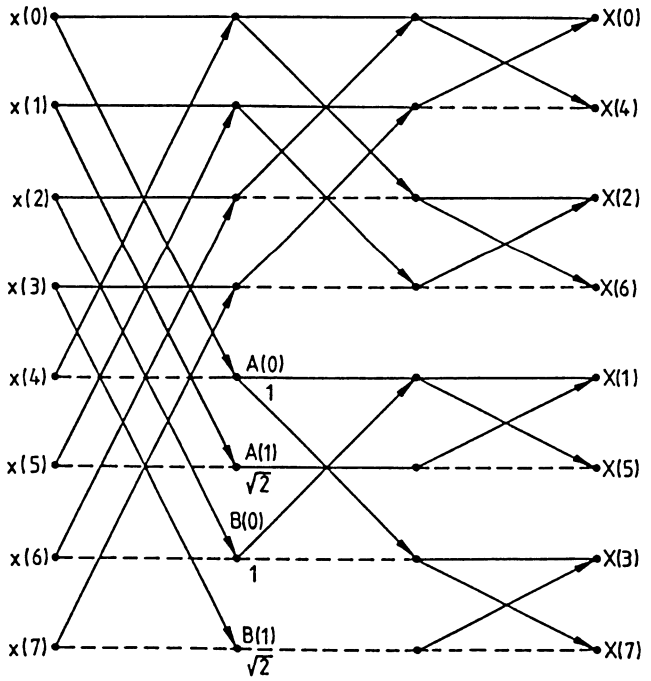


Bild 4.20. Vollständiger Split-Radix-SFG (N=8) der DHYT, nach [4.21]

4.3.5 Berechnung der DHYT aus Koeffizienten von Teilfolgen

In diesem Abschnitt betrachten wir einen schnellen Algorithmus der Hartley-Transformation, dessen allgemeine Strategie im Abschnitt 3.1.2 erläutert wurde, nämlich die Berechnung aus den Koeffizienten niedrigerer Ordnung.

Die rekursive Struktur des schnellen Hartley-Transformationsalgorithmus' ermöglicht eine Erzeugung der nächsthöheren Ordnung der DHYT aus zwei identischen DHYT niedrigerer Ordnung [4.22]. Als Beispiel werden wir den Fall N=8 auf diese Weise diskutieren.

Die Matrixdarstellung der DHYT lautet:

$$\begin{bmatrix} X_{HY}(0) \\ X_{HY}(1) \\ X_{HY}(2) \\ \vdots \\ X_{HY}(N-1) \end{bmatrix} = N^{-1/2} \begin{matrix} k \\ \| \\ 0 \\ 1 \\ 2 \\ \vdots \\ N-1 \end{matrix} \begin{bmatrix} & & \\ & \text{cas}(2\pi kn/N) & \\ & & \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{bmatrix}, \qquad n = 0,1,\ldots,N-1.$$

Zur Vereinfachnung lassen wir den normalisierenden Faktor $N^{-1/2}$ weg. Wir betrachten die Fälle N=2 bis N=8:

(1) DHYT erster Ordnung (r=1, N=2):

$$\begin{bmatrix} X_{HY}(0) \\ X_{HY}(1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \end{bmatrix} \qquad \text{d.h. } [HY(1)] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Die Matrix [HY(1)] entspricht (wie bei allen orthogonalen Transformationen) der Hadamard-Matrix der ersten Ordnung. Den Signalflußgraph für [HY(1)] zeigt das Bild 4.21(a).

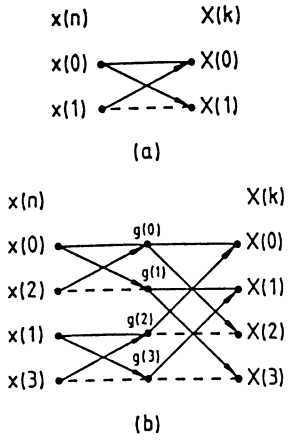


Bild 4.21. Signalflußgraph für die DHYT,
a) N=2, b) N=4, nach [4.22]

(2) DHYT zweiter Ordnung (r=2, N=4):

$$\begin{bmatrix} X_{HY}(0) \\ X_{HY}(1) \\ \\ X_{HY}(2) \\ X_{HY}(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ & & & \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ \\ x(2) \\ x(3) \end{bmatrix}, \tag{4.74}$$

d.h.

$$[HY(2)] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ & & & \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}.$$

Nach einer Spaltenumordnung in (4.74) erhält man

$$\begin{bmatrix} X_{HY}(0) \\ X_{HY}(1) \\ \\ X_{HY}(2) \\ X_{HY}(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ & & & \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(2) \\ \\ x(1) \\ x(3) \end{bmatrix}.$$

Mit $2g(0) = x(0) + x(2)$, $2g(1) = x(0) - x(2)$,
$2g(2) = x(1) + x(3)$, $2g(3) = x(1) - x(3)$

erhält man die DHYT-Koeffizienten $\underline{X}_{HY}$ unter Verwendung der entsprechend modifizierten Matrix [HY(2)]':

$$\begin{bmatrix} X_{HY}(0) \\ X_{HY}(1) \\ \\ X_{HY}(2) \\ X_{HY}(3) \end{bmatrix} = [HY(2)]'\ g_4 = \left[\begin{array}{cc|cc} 1 & & 1 & \\ & 1 & & 1 \\ \hline 1 & & -1 & \\ & 1 & & -1 \end{array}\right] \begin{bmatrix} g(0) \\ g(1) \\ -- \\ g(2) \\ g(3) \end{bmatrix}.$$

Man beachte, daß die Daten in Bitumkehrfolge angeordnet sind. Den entsprechenden Signalflußgraph zeigt das Bild 4.21(b).

(3) DHYT 3. Ordnung (r=3, N=8):

$$\begin{bmatrix} X_{HY}(0) \\ X_{HY}(1) \\ X_{HY}(2) \\ X_{HY}(3) \\ \\ X_{HY}(4) \\ X_{HY}(5) \\ X_{HY}(6) \\ X_{HY}(7) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \sqrt{2} & 1 & 0 & -1 & -\sqrt{2} & -1 & 0 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 0 & -1 & \sqrt{2} & -1 & 0 & 1 & -\sqrt{2} \\ & & & & & & & \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -\sqrt{2} & 1 & 0 & -1 & \sqrt{2} & -1 & 0 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 0 & -1 & -\sqrt{2} & -1 & 0 & 1 & \sqrt{2} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix}.$$

Eine Spaltenumordnung führt zu

$$\begin{bmatrix} X_{HY}(0) \\ X_{HY}(1) \\ X_{HY}(2) \\ X_{HY}(3) \\ --- \\ X_{HY}(4) \\ X_{HY}(5) \\ X_{HY}(6) \\ X_{HY}(7) \end{bmatrix} = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 0 & 0 & \sqrt{2} & -\sqrt{2} \\ \hline 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -\sqrt{2} & \sqrt{2} & 0 & 0 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 0 & 0 & -\sqrt{2} & \sqrt{2} \end{array}\right] \begin{bmatrix} x(0) \\ x(4) \\ x(2) \\ x(6) \\ \\ x(1) \\ x(5) \\ x(3) \\ x(7) \end{bmatrix} =$$

$$
= \left[\begin{array}{cccc|cccc}
1 & & & & 1 & & & \\
 & 1 & & & & \alpha & & \alpha \\
 & & 1 & & & & 1 & \\
 & & & 1 & & \alpha & & -\alpha \\
\hline
1 & & & & -1 & & & \\
 & 1 & & & & -\alpha & & -\alpha \\
 & & 1 & & & & -1 & \\
 & & & 1 & & -\alpha & & \alpha
\end{array}\right]
\left[\begin{array}{c} g(0) \\ g(1) \\ g(2) \\ g(3) \\ \hline g(4) \\ g(5) \\ g(6) \\ g(7) \end{array}\right] . \qquad (4.75)
$$

Dabei ist $\alpha = 1/\sqrt{2}$ und die g_N sind DHYT-Koeffizienten der Ordnung 2. Der zugehörige SFG ist im Bild 4.22 dargestellt.

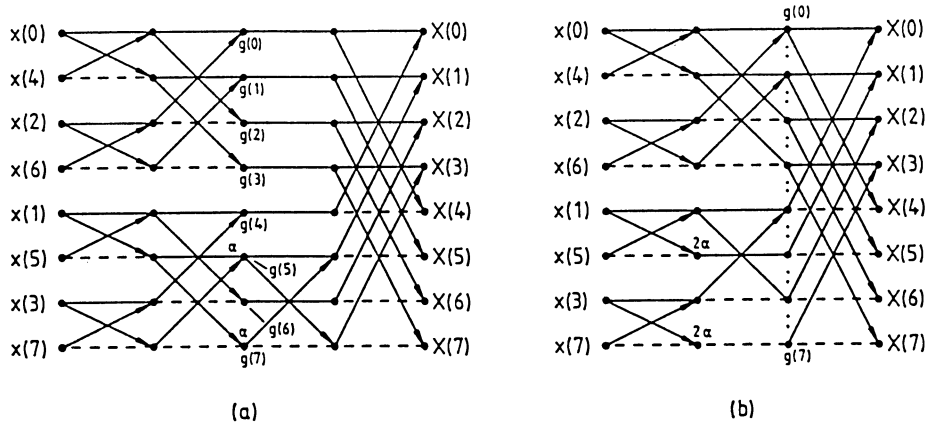


Bild 4.22. Signalflußgraph für die DHYT, N=8, a) SFG entspr. (4.75), b) vereinfachter SFG, nach [4.22]

Aus diesen Beispielen ist ersichtlich, daß die Transformationsmatrix [HY(L)] offenbar rekursiv unter Verwendung der Matrix [HY(L-1)] erzeugt werden kann [4.22]. Dazu wird eine Umordnung von Zeilen und Spalten vorgenommen, mit der die Transformationsmatrix in 4 Quadranten zerlegt wird:

$$
[HY(L)] = \left[\begin{array}{c|c} [Q_1(L-1)] & [Q_3(L-1)] \\ \hline [Q_2(L-1)] & [Q_4(L-1)] \end{array}\right] .
$$

Wenn man die geradzahlig (ungeradzahlig) indizierten Zeilen in der oberen (unteren) Hälfte der Matrix anordnet, folgt aus den Eigenschaften der Cosinus- und Sinus-Funktion, daß

$$[Q_1(L)] = [Q_2(L)] = [HY(L-1)] \text{ und } [Q_4(L)] = -[Q_3(L)].$$

Für die Teilmatrizen $[Q_3]$ und $[Q_4]$ ergibt die Herleitung [4.22]

$$[Q_3(L-1)] = [HY'(L-1)][DI(L-1)],$$

wobei $[HY'(L-1)]$ die gemäß dem Gray-Code umgeordnete Matrix $[HY(L-1)]$ ist. $[DI(L-1)]$ ist gegeben durch

$$[DI(L-1)] = [Diag(\cos(2\pi k/N))] + [Diag(\sin(2\pi k/N))][P],$$

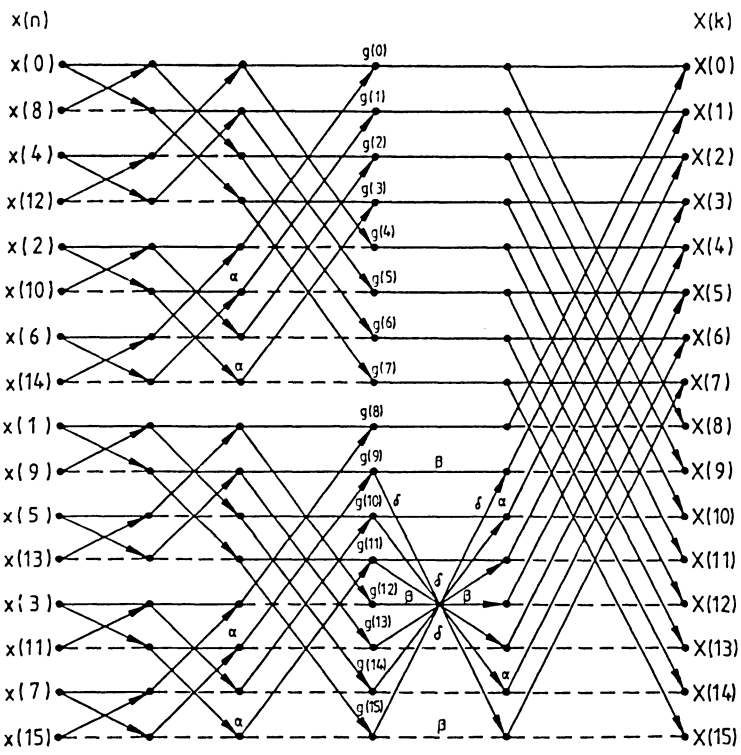


Bild 4.23. Signalflußgraph für die DHYT, N=16, nach [4.22]

mit der Permutationsmatrix

$$[P] = \begin{bmatrix} 1 & & & & \\ & & & 1 & \\ & & 1 & & \\ & \cdot & & & \\ & & \cdot & & \\ & 1 & & & \\ 1 & & & & \end{bmatrix}.$$

Damit folgt für den DHYT-Algorithmus (mit Zeitdezimierung)

$$\begin{bmatrix} X_{HY}(0 \ \ldots \ (N/2)-1) \\ - - - - - - - - - - - \\ X_{HY}((N/2) \ \ldots \ N-1) \end{bmatrix} = N^{-1} \begin{bmatrix} [HY'^{T}(L-1)] & [DI(L-1)][HY'^{T}(L-1)] \\ - - - - - - & - - - - - - - - - - - \\ [HY'^{T}(L-1)] & [DI(L-1)][HY'^{T}(L-1)] \end{bmatrix} \begin{bmatrix} x_g \\ \\ x_u \end{bmatrix}. \quad (4.76)$$

Als weiteres Beispiel betrachten wir die DHYT für N=16. Die nach (4.76) entwickelte Matrix [HY(3)] kann (analog zu (4.75)) rekursiv dargestellt werden. Ihre Umsetzung als SFG ist im Bild 4.23 angegeben. Dabei bedeuten: $\alpha = 1/\sqrt{2}$, $\beta = \cos(\pi/8)$ und $\delta = \sin(\pi/8)$.

4.3.6 DHYT-Berechnung aus Walsh-Koeffizienten

Im Abschnitt 3.1.3 wurde die Berechnung einer Transformation aus Koeffizienten einer anderen Transformation allgemein erläutert. In diesem Abschnitt diskutieren wir die schnelle Berechnung der Hartley-Transformation durch die WHT [4.25], [4.26]. Das Verfahren läuft in einem (N×N)-Datenblock, wobei $N=2^r$. Die Walsh-Hadamard-Koeffizienten werden direkt erzeugt und dann zur Gewinnung der DHYT-Koeffizienten verwendet. Dies geschieht durch eine Transformationsmatrix [G], die orthonormal ist, und eine blockdiagonale Struktur aufweist.

Zuerst betrachten wir die Herleitung der Matrix [G]. Seien $\underline{X}_{HY}$ und $\underline{X}_{WH}$ Spaltenvektoren, nämlich die Koeffizientenvektoren der Hartley- bzw. der Walsh-Hadamard-Transformierten. Dann gilt für die Beziehung zwischen $\underline{X}_{HY}$ und $\underline{X}_{WH}$

$$\underline{X}_{HY<k>} = [G(L)]\, \underline{X}_{WHk}. \quad (4.77)$$

Damit ist $[G(L)] = [HY(<L>)][H_H(L)]$. Dabei ist $[HY(<L>)]$ die (N×N)-Transformationsmatrix der DHYT mit Bitumkehr von L und $[H_H(L)]$ die (N×N)-Transformationsmatrix der WHT.

Interessant ist dabei die Tatsache, daß die Matrix [G] orthonormal ist und eine blockdiagonale Struktur aufweist. Damit wird z.B. eine Parallelverarbeitung in VLSI erleichtert.

Nun betrachten wir den Transformationskern

$$cas(2\pi nk/N) = a_{nk} = \cos(2\pi nk/N) + \sin(2\pi nk/N),$$

d.h. die Transformationsmatrix $[HY(L)] = [a_{nk}]$. Wir ordnen nun die Zeilenreihenfolge von [HY(L)] durch Bitumkehr um, und setzen

$$[A(L)] = [HY'(L)] = [b_{<n>k}].$$

Nun sei $n = (n_{r-1}...n_0) = n_{r-1}2^{r-1} + ... + n_0 2^0$

mit $r = \log_2 N$, $0 \leq n \leq N-1$, $n_{r-1}, n_{r-2}, ..., n_0 \in \{0,1\}$.

Wenn $n_0 = 0$, dann erhalten wir eine gerade Zahl n. Somit ist

$$\begin{aligned} b_{<n>(k+(N/2))} &= a_{n(k+(N/2))} = cas(2\pi n(k+N/2)/N) \\ &= cas((2\pi nk/N)+\pi n) = cas(2\pi nk/N) = a_{nk} = b_{<n>k}. \end{aligned} \qquad (4.78)$$

Wenn andererseits $k_0 = 1$, dann ist k eine ungerade ganze Zahl. Damit folgt

$$\begin{aligned} b_{<n>(k+(N/2))} &= a_{n(k+(N/2))} = cas(2\pi n(k+(N/2))/N) = cas((2\pi nk/N)+\pi n) \\ &= - cas(2\pi nk/N) = - a_{nk} = - b_{<n>k}. \end{aligned} \qquad (4.79)$$

Mit (4.78) und (4.79) können wir [A(L)] rekursiv schreiben

$$[A(L)] = \begin{bmatrix} [A(L-1)] & [A(L-1)] \\ [B(L-1)] & -[B(L-1)] \end{bmatrix}. \qquad (4.80)$$

Gleichung (4.80) zeigt, daß die (N×N)-Matrix [A(L)] in zwei zerlegbare Untermatrizen [A(L-1)] und zwei unzerlegbare Untermatrizen [B(L-1)] der Ordnung (N/2)×(N/2) aufgeteilt werden kann. Man kann dann [A(L-1)] in (N/4)×(N/4)-Untermatrizen [A(L-2)] und [B(L-2)] zerlegen. Dieses iterative Verfahren endet, wenn

$$[A(2)] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ & & & \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}. \tag{4.81}$$

Für die rekursive Entwicklung der Hadamard-Matrix gilt

$$[H_H(L)] = \begin{bmatrix} [H_H(L-1)] & [H_H(L-1)] \\ & \\ [H_H(L-1)] & -[H_H(L-1)] \end{bmatrix}$$

mit $[H_H(2)] = [A(2)]$. Aus (4.77) erhält man

$$[G(L)] = [HY'(L)][H_H(L)] = N^{-1}\begin{bmatrix} [A(L-1)] & [A(L-1)] \\ & \\ [B(L-1)] & -[B(L-1)] \end{bmatrix}\begin{bmatrix} [H_H(L-1)] & [H_H(L-1)] \\ & \\ [H_H(L-1)] & -[H_H(L-1)] \end{bmatrix}$$

$$= N^{-1}\begin{bmatrix} 2[A(L-1)][H_H(L-1)] & [A(L-1)]([H_H(L-1)]-[H_H(L-1)]) \\ & \\ [B(L-1)]([H_H(L-1)]-[H_H(L-1)]) & 2[B(L-1)][H_H(L-1)] \end{bmatrix}$$

$$= 2N^{-1}\begin{bmatrix} 2[A(L-1)][H_H(L-1)] & 0 \\ & \\ 0 & 2[B(L-1)][H_H(L-1)] \end{bmatrix}.$$

Nach $\log_2 N$ Iterationen wird die o.g. Diagonalmatrix erreicht

$$[A(2)][H_H(2)] = 4\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ & & & \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

So findet man die folgende allgemeine Struktur der Matrix [G]:

$$[G(r)] = \left[\begin{array}{cccc|cccccccccccccccc}
1 & 0 & 0 & 0 & & & & & & & & & & & & & & \\
0 & 1 & 0 & 0 & & & & & & & & & & & & & & \\
0 & 0 & 1 & 0 & & & & & & & & & & & & & & \\
0 & 0 & 0 & 1 & & & & & & & & & & & & & & \\
\hline
 & & & & x & y & z & t & & & & & & & & & & \\
 & & & & y & x & t & z & & & & & & & & & & \\
 & & & & z & t & y & x & & & & & & & & & & \\
 & & & & t & z & x & y & & & & & & & & & & \\
\hline
 & & & & & & & & a & b & c & d & e & f & g & h \\
 & & & & & & & & b & a & d & c & f & e & h & g \\
 & & & & & & & & c & d & b & a & h & g & e & f \\
 & & & & & & & & d & c & a & b & g & h & f & e \\
 & & & & & & & & -g & -h & -f & -e & d & c & b & a \\
 & & & & & & & & -h & -g & -e & -f & c & d & a & b \\
 & & & & & & & & -f & -e & -g & -h & b & a & d & c \\
 & & & & & & & & -e & -f & -h & -g & a & b & c & d
\end{array}\right] .$$

Als Beispiel betrachten wir den Fall r=3 (N=8):

$$[G(3)] = \left[\begin{array}{cccc|cccccccc}
1 & 0 & 0 & 0 & & & & & & & & \\
0 & 1 & 0 & 0 & & & & & 0 & & & \\
0 & 0 & 1 & 0 & & & & & & & & \\
0 & 0 & 0 & 1 & & & & & & & & \\
\hline
 & & & & & & & & & & & \\
 & & & & & & & & & & & \\
 & & & & & & & & & & & \\
 & 0 & & & & & & & (1/4)[B(2)][H_H(2)] & & & \\
 & & & & & & & & & & &
\end{array}\right] = (1/8)[HY'(3)][H_H(3)].$$

Weil $[A(L)] = [b_{<n>k}] = [a_{nk}]$ und $a_{nk} = \cos(2\pi nk/N)+\sin(2\pi nk/N)$ ist, wird

$$[A(3)] = \left[\begin{array}{cccc|cccc} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} & a_{06} & a_{07} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{60} & a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ \hline a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{50} & a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{70} & a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{array}\right]$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ & & & & & & & \\ 1 & \sqrt{2} & 1 & 0 & -1 & -\sqrt{2} & -1 & 0 \\ 1 & -\sqrt{2} & 1 & 0 & -1 & \sqrt{2} & -1 & 0 \\ 1 & 0 & -1 & \sqrt{2} & -1 & 0 & 1 & -\sqrt{2} \\ 1 & 0 & -1 & -\sqrt{2} & -1 & 0 & 1 & \sqrt{2} \end{bmatrix}.$$

Aus (4.80) folgt

$$[B(2)] = \begin{bmatrix} 1 & \sqrt{2} & 1 & 0 \\ 1 & -\sqrt{2} & 1 & 0 \\ & & & \\ 1 & 0 & -1 & \sqrt{2} \\ 1 & 0 & -1 & -\sqrt{2} \end{bmatrix}$$

und

$$4[B(2)][H_H(2)] =$$

$$\begin{bmatrix} 2+\sqrt{2} & 2-\sqrt{2} & \sqrt{2} & -\sqrt{2} \\ 2-\sqrt{2} & 2+\sqrt{2} & -\sqrt{2} & \sqrt{2} \\ & & & \\ \sqrt{2} & -\sqrt{2} & 2-\sqrt{2} & 2+\sqrt{2} \\ -\sqrt{2} & \sqrt{2} & 2+\sqrt{2} & 2-\sqrt{2} \end{bmatrix} = \begin{bmatrix} 0{,}854 & 0{,}146 & 0{,}354 & -0{,}354 \\ 0{,}146 & 0{,}854 & -0{,}354 & 0{,}354 \\ & & & \\ 0{,}354 & -0{,}354 & 0{,}146 & 0{,}854 \\ -0{,}354 & 0{,}354 & 0{,}854 & 0{,}146 \end{bmatrix}.$$

Es ergibt sich deshalb

$$[G(3)] = \left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & & & & \\ 0 & 1 & 0 & 0 & & & & \\ 0 & 0 & 1 & 0 & & & & \\ 0 & 0 & 0 & 1 & & & & \\ \hline & & & & 0{,}854 & 0{,}146 & 0{,}354 & -0{,}354 \\ & & & & 0{,}146 & 0{,}854 & -0{,}354 & 0{,}354 \\ & & & & 0{,}354 & -0{,}354 & 0{,}146 & 0{,}854 \\ & & & & -0{,}354 & 0{,}354 & 0{,}854 & 0{,}146 \end{array}\right].$$

Wenn die $(WHT)_H$-Koeffizienten $X_{WH}(k)$ bekannt sind, lassen sich nun die DHYT-Koeffizienten gemäß (4.77) mit Hilfe der Konversionsmatrix [G] berechnen.

4.3.7 Konversion zwischen DFT- und DHYT-Koeffizienten

Bereits im vorhergehenden Abschnitt haben wir ein Beispiel dafür gesehen, wie Koeffizienten einer Transformation in die entsprechenden einer anderen Transformation umgerechnet werden können. Nun wollen wir die Konversion von Fourier-Koeffizienten in Hartley-Koeffizienten betrachten. Das ist besonders interessant, weil beide Transformationen eng verwandt sind.

Jede FFT kann durch Veränderung der Indizierung in eine schnelle Hartley-Transformation konvertiert werden [4.27]. Wie in den vorhergehenden Abschnitten erwähnt wurde, kann man die schnelle DHYT eines reellwertigen N-Punkte-Datenverktors in der halben Zeit (verglichen mit der FFT) ausführen. Es bietet sich daher an, die DFT über die DHYT zu berechnen, zumal sich die Konvertierung problemloser gestaltet als die optimale Nutzung der DFT für reelle Daten. Formal ist die Beziehung zwischen der DFT und der DHYT aus ihren Definitionen ersichtlich:

$$X_{HY}(k) = \sum_{n=0}^{N-1} x(n)\ \{\cos(2\pi kn/N) + \sin(2\pi kn/N)\},$$

$$X_f(k) = \sum_{n=0}^{N-1} x(n)\ \{\cos(2\pi kn/N) - j\ \sin(2\pi kn/N)\}.$$

Wie aus Abschnitt 2.2 folgt, existieren die folgenden Beziehung zwischen Fourier-Transformierten und Hartley-Transformierten:

$$X_f(k) = (X_{HY}(k)+jX_{HY}(-k))/(1+j), \tag{4.82}$$

$$X_{HY}(k) = \mathrm{Re}\{(1+j)X_f(k)\} \quad \text{oder} \quad X_{HY}(-k) = \mathrm{Im}\{(1+j)X_f(k)\}.$$

Dabei wurde die Identität

$$\exp(-j2\pi kn) = [(1+j)\cos(2\pi kn)+(1-j)\sin(2\pi kn)]/(1+j)$$

benutzt. Hieraus folgt aufgrund der Symmetrieeigenschaften der Fourier-Matrix

$$X_{HY} = \mathrm{Re}\{X_f\} - \mathrm{Im}\{X_f\}.$$

Ein Vergleich der beiden Transformationen zeigt, daß die Theoreme der Fourier-Transformation auf die Hartley-Transformation übertragen werden können, wobei das Auftreten der imaginären Einheit j bei der DFT mit einem Vorzeichenwechsel des Arguments (z.B. k —> -k) bei der DHYT korrespondiert. Transformationen über 2N Punkte werden wir in den folgenden Gleichungen durch das Zeichen "~" bezeichnen. Die Summation wird dabei in zwei separate Summen mit gerader bzw. ungerader Indizierung zerlegt

$$\tilde{X}_f(k) = \sum_{m=0}^{2N-1} x(m)\exp(-j2\pi km/(2N))$$

$$= \sum_{n=0}^{N-1} x(2n)\exp(-j2\pi kn/N) + \exp(-j\pi k/N)\sum_{n=0}^{N-1} x(2n+1)\exp(-j2\pi kn/(2N)).$$

Aus dieser Form ersieht man, daß die beiden Summationen die N-Punkte-Fourier-Transformierten der gerade bzw. der ungerade indizierten Daten sind. Diese werden als X_{fg} bzw. X_{fu} bezeichnet

$$\tilde{X}_f(k) = X_{fg}(k) + \exp(-j\pi k/N)X_{fu}(k).$$

Mit (4.82) erhalten wir

$$\tilde{X}_{HY}(k) + j\tilde{X}_{HY}(-k) = X_{HYg}(k) + jX_{HYg}(-k) + \exp(-j\pi k/N)\{X_{HYu}(k)+jX_{HYu}(-k)\}. \tag{4.83}$$

Gleichung (4.83) zeigt, daß ein $N\log_2 N$-Algorithmus sowohl für die komplexe als auch für reelle diskrete Hartley-Transformation existiert. Nun betrachten wir lediglich den Realteil von (4.83) und erhalten

$$\tilde{X}_{HY}(k) = X_{HYg}(k) + \cos(\pi k/N)X_{HYu}(k) + \sin(\pi k/N)X_{HYu}(-k).$$

Der imagnäre Teil beschreibt dieselbe Beziehung jedoch mit "-k" anstatt "+k". Die Analogien zwischen Fourier- und Hartley-Transformation machen letztere zu einer interessanten Alternative für viele Anwendungen. Ein FORTRAN-Programm zur schnellen Berechnung der DHYT ist z.B. in [4.27] angegeben.

4.4 Algorithmen für die diskrete Cosinus-Transformation

Die Algorithmen für die diskrete Cosinus-Transformation (DCT) basieren auf unterschiedlichen Anzätzen: Einige Algorithmen (vgl. die Abschnitte 4.4.1 bis 4.4.3) nutzten die Verwandtschaft der DCT mit der DFT oder der DHYT aus und verwenden deren Algorithmen. Im allgemeinen ist es aber effizienter, die DCT direkt zu berechnen (Abschnitte 4.4.4 und 4.4.5). Die Abschnitte 4.4.6 und 4.4.7 sind speziellen Verfahren, nämlich der Berechnung aus Teilfolgen und einer modifizierten DCT gewidmet, schließlich sei auch erwähnt, daß für die DCT auch ein Primfaktor-Algorithmus existiert [4.33].

Die konventionelle Methode zur Ausführung einer N-Punkte-DCT benutzt den 2N-Punkte-DFT-Algorithmus [4.28] und [4.29], der komplexe Arithmetik überall in der Berechnung verwendet. Dabei gilt

$$X_c(0) = \sqrt{2}\sum_{n=0}^{N-1} x(n),$$

$$X_c(k) = 2\sum_{n=0}^{N-1} x(n)\cos((2n+1)k\pi/(2N)) \qquad k = 1,2,\ldots,N-1.$$

Der Umweg über die DFT und die damit verbundenen komplexen Basisfunktionen lassen diesen Weg allerdings nur in Ausnahmefällen als geeignet erscheinen. Eine Einsparung von etwa 50% des Rechenaufwandes erzielt man, wenn man die Symmetrieeigenschaften der DFT geeignet ausnutzt (vgl. Abschnitt 4.2.7).

4.4.1 DCT-Berechnung mittels 2N-Punkte-DFT

In diesem Abschnitt wird ein N-Punkte-DCT-Algorithmus diskutiert, der von der 2N-Punkte-DFT abgeleitet ist. Obwohl dem Fall $N=2^r$ ein besonderes Interesse gilt, sei erwähnt, daß dieser DCT-Algorithmus für beliebiges N verallgemeinert werden kann, wobei N ungerade oder gerade sein kann [4.28]. Außerdem besteht die Möglichkeit der Erweiterung der Dimension zum m-dimensionalen Fall.

Die Cosinus-Transformierte einer diskreten zeitlichen Folge x(n) ist entsprechend ihrer Definition der reelle Teil ihrer Fourier-Transformierten $X_f(k)$. Der reelle Teil von $X_f(k)$ ist gleich der Fourier-Transformierten des geraden Teils von x(n), der durch $x_g(n) = [x(n)+x(-n)]/2$ definiert ist. Deshalb ist die Cosinus-Transformierte von x(n) gleich der Fourier-Transformierten von $x_g(n)$. Wenn x(n) kausal ist, d.h. $x(n)=0$ für $n<0$, spezifiziert die Cosinus-Transformierte von $x_g(n)$ die Folge x(n) eindeutig. In diesem Fall kann $x_g(n)$ als eine gerade Erweiterung von x(n) angesehen werden. Deshalb läßt sich die Cosinus-Transformierte einer kausalen Folge x(n) aus der Fourier-Transformierten einer geraden Erweiterung von x(n) gewinnen.

Bild 4.24(a) zeigt ein kausales Signal x(n) und Bild 4.24(b) eine Erweiterung von x(n):

$$y_1(n) = x(n) + x(-n) = 2 \cdot x_g(n).$$

Eine andere mögliche gerade Erweiterung von x(n) wird durch Bild 4.24(c) illustriert, wobei $y_2(n) = x(n) + x(-n-1)$. $y_1(n)$ ist gerade bezogen auf den Punkt n=0, während $y_2(n)$ gerade zum Punkt n=-0.5 ist. Die Fourier-Transformierte von y_1 ist reell, während die Fourier-Transformierte von $y_2(n)$ einen der Verschiebung entsprechenden linearen Phasenanteil enthält. Die auf $y_1(n)$ oder $y_2(n)$ basierenden Cosinus-Transformationen können entsprechend definiert werden. Dabei kann eindeutig entschieden werden, aus welchem x(n) sie entstanden.

In diesem Beispiel haben wir angenommen, daß die Fourier-Transformierte für alle Frequenzen berechnet wird. In Wirklichkeit berechnet man die DFT für eine endliche Anzahl von Frequenzen gleichen Abstands. Wenn man versucht,

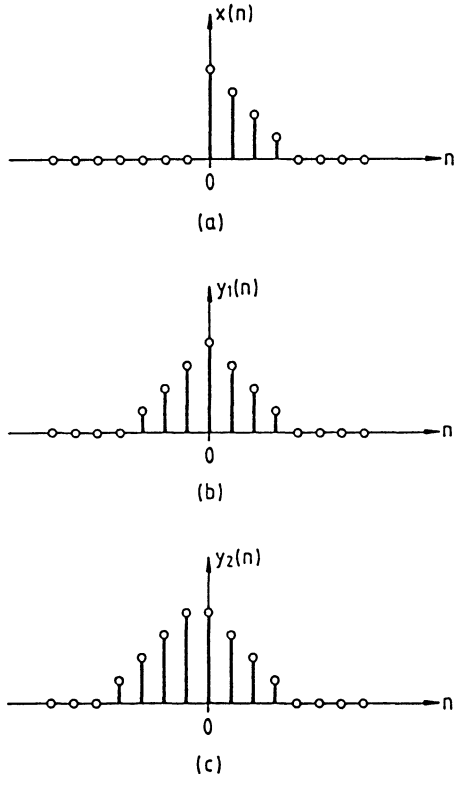


Bild 4.24. a) Kausales Signal x(n), b) Eine gerade Erweiterung von x(n), $y_1(n)$ symmetrisch zu n=0, c) Eine andere gerade Erweiterung von x(n), $y_2(n)$ symmetrisch zu n = −0,5, nach [4.28]

eine gerade Erweiterung eines Signals zu formulieren, wobei die erweiterten Signale periodisch sein müssen, hat man eine zusätzliche Wahl in der Definition der Erweiterungen.

Als Beispiel mit N=4 sei x(n) ($0 \leq n \leq 3$) eine durch die vier von Null verschiedenen Abtastungen der im Bild 4.24(a) gegebene Sequenz. Bild 4.25 zeigt vier verschiedene gerade Erweiterungen von x(n). $y_1(n)$ ist eine gerade (2N−2)-Erweiterung, während $y_2(n)$ und $y_3(n)$ zwei unterschiedliche gerade (2N−1)-Erweiterungen sind. $y_4(n)$ ist eine gerade (2N)-Erweiterung. Jede der vier Erweiterungsdefinitionen könnte die Basis für eine DCT-Definition bilden. Die üblichste Form der DCT ist die aus der geraden 2N-Erweiterung $y_4(n)$ abgeleitete Form sowie die im folgenden diskutierte Form.

Die DCT von x(n) ist aus der DFT der geraden 2N-Erweiterung des Signals x(n) herzuleiten. Sei y(n) eine gerade 2N-Erweiterung von x(n) und wie folgt definiert

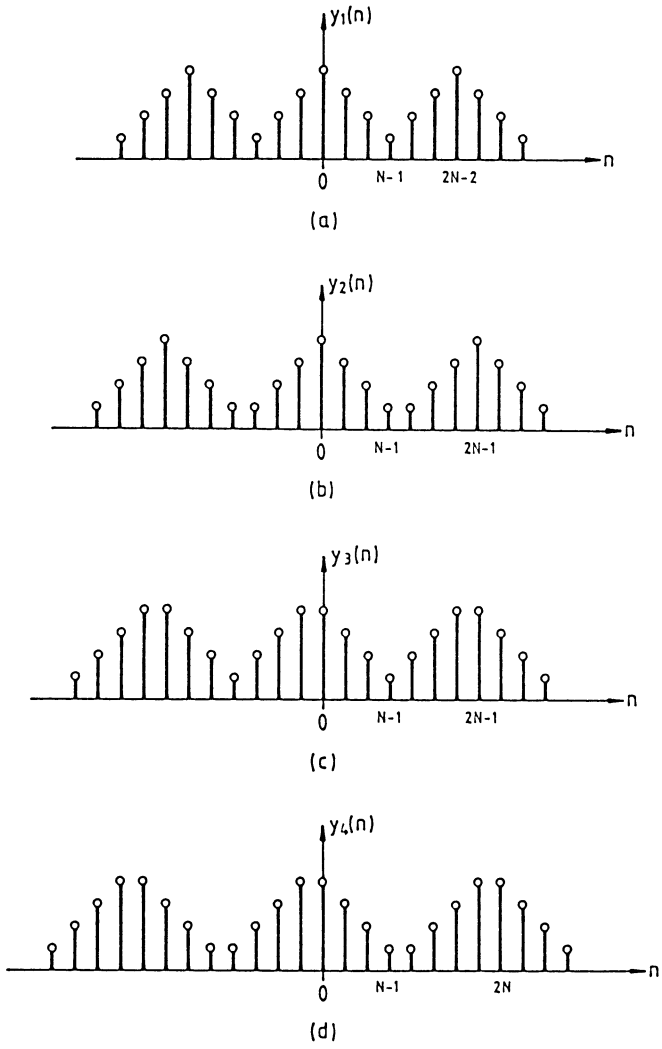


Bild 4.25. Vier verschiedene periodische gerade Erweiterungen von nicht verschwindenden Abtastungen x(n), nach [4.28]

$$y(n) = \begin{cases} x(n) & 0 \leq n \leq N-1 \\ x(2N-n-1) & N \leq n \leq 2N-1, \end{cases} \tag{4.84}$$

dann ist $y(2N-n-1) = y(n)$. (4.85)

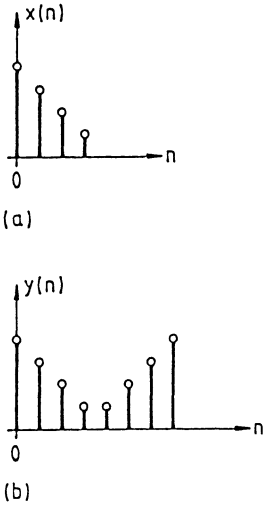


Bild 4.26. a) Ursprüngliches Signal x(n), $0 \leq n \leq N-1$, b) Eine gerade 2N-Punkte-Erweiterung von x(n), y(n), nach [4.28]

Das Bild 4.26(a) zeigt ein Beispiel eines Signals x(n) und Bild 4.26(b) die entsprechende gerade Erweiterung von x(n) wie in (4.84) definiert. Wegen der "-1" auf der linken Seite von (4.85) ist y(n) nicht gerade zu N und wird deshalb keine reelle DFT haben.

Die DFT von y(n) ist gegeben durch

$$Y_f(k) = \sum_{n=0}^{2N-1} y(n)\, W_{2N}^{nk}. \tag{4.86}$$

Substitution von (4.84) in (4.86) führt zu

$$Y_f(k) = \sum_{n=0}^{N-1} x(n)\, W_{2N}^{nk} + \sum_{n=N}^{2N-1} x(2N-n-1)\, W_{2N}^{nk}.$$

Nach Substitution der Summationsvariablen auf der rechten Seite und unter Beachtung von $W_{2N}^{2nN}=1$ für ganze Zahlen n sowie Ausklammern von $W_{2N}^{(-k/2)}$, erhält man

$$Y_f(k) = W_{2N}^{(-k/2)} \sum_{n=0}^{N-1} x(n)\, (W_{2N}^{(nk+(k/2))} + W_{2N}^{-(nk-(k/2))}). \tag{4.87}$$

Die Darstellung in (4.87) kann auf zwei verschiedene Weisen geschrieben werden, nämlich

$$Y_f(k) = W_{2N}^{(-k/2)} 2 \sum_{n=0}^{N-1} x(n)\, \cos((2n+1)k\pi/(2N)). \tag{4.88}$$

oder

$$Y_f(k) = W_{2N}^{(-k/2)} 2 \cdot \mathrm{Re}\{W_{2N}^{(k/2)} \sum_{n=0}^{N-1} x(n)\ W_{2N}^{nk}\}. \qquad 0 \leq k \leq 2N-1 \tag{4.89}$$

Aus der Definition der DCT von x(n)

$$X_c(k) = 2 \sum_{n=0}^{N-1} x(n)\ \cos((2n+1)k\pi/(2N)), \qquad 0 \leq k \leq N-1 \tag{4.90}$$

und (4.88) ergibt sich

$$X_c(k) = W_{2N}^{(k/2)} Y_f(k) \text{ oder } Y_f(k) = W_{2N}^{-(k/2)} X_c(k). \tag{4.91}$$

Mit (4.89) und (4.91) folgt dann

$$X_c(k) = 2 \cdot \mathrm{Re}\{W_{2N}^{(k/2)} \sum_{n=0}^{N-1} x(n)\ W_{2N}^{nk}\}. \tag{4.92}$$

Die Gleichungen (4.91) spezifizieren die Beziehung zwischen der DCT einer Sequenz und der DFT der 2N-Erweiterung dieser Sequenz. $X_c(k)$ ist reell, während $Y_f(k)$ komplex ist. Die DCT von x(n) läßt sich durch Berechnung der 2N-Punkte-DFT von y(n) gemäß (4.86) und Multiplikation des Ergebnisses mit $W_{2N}^{k/2}$ ermitteln.

Aus (4.92) sieht man, daß sich die DCT auch berechnen läßt, indem die 2N-Punkte-DFT der ursprünglichen Sequenz x(n) mit dazugefügten N Nullen berechnet wird, die Ergebnisse mit $W_{2N}^{k/2}$ multipliziert werden und hiervon der verdoppelte Realteil genommen wird.

Nun betrachte man die inverse DCT (IDCT), abgeleitet aus der inversen DFT (IDFT). Die IDFT von $Y_f(k)$ ist gegeben durch

$$y(n) = (2N)^{-1} \sum_{n=0}^{N-1} Y_f(k)\ W_{2N}^{-kn}.$$

Weil y(n) reell ist, ist $Y_f(k)$ hermitisch symmetrisch, d.h.

$$Y_f(2N-k) = Y_f^*(k). \tag{4.93}$$

Außerdem gilt nach (4.91): $Y_f(N) = 0.$ (4.94)

Unter Verwendung von (4.93), (4.94) und (4.84) ergibt sich

$$y(n) = N^{-1}\ \mathrm{Re}\{Y_f(0)/2 + \sum_{k=0}^{2N-1} X_c(k)\ \cos((2n+1)k\pi/(2N))\} \qquad 0 \leq n \leq 2N-1. \quad (4.95)$$

Durch Einsetzen von (4.91) in (4.95) und nach Anwendung von (4.84) erhält man für die IDCT

$$x(n) = N^{-1}\{\ X_c(0)/2 + \sum_{k=1}^{N-1} X_c(k)\ \cos((2n+1)k\pi/(2N))\}, \qquad 0 \leq n \leq N-1.$$

Die Gleichungen (4.90) und (4.95) bilden ein DCT-Paar, d.h.

$$X_c(k) = 2\sum_{n=0}^{N-1} x(n)\ \cos((2n+1)k\pi/(2N)), \qquad 0 \leq k \leq N-1,$$

$$\updownarrow$$

$$x(n) = N^{-1}\ \{X_c(0)/2 + \sum_{k=0}^{N-1} X_c(k)\cos((2n+1)k\pi/(2N))\}, \qquad 0 \leq n \leq N-1.$$

Sei $X_c(k)$ gegeben, so wird $x(n)$ zuerst durch Berechnung von $Y_f(k)$ unter Verwendung von (4.91) und dann durch Anwendung der durch (4.95) gegebenen komplexen 2N-Punkte-IDFT erhalten, was zu $y(n)$ und folglich $x(n)$ führt.

4.4.2 DCT-Berechnung mittels N-Punkte-DFT

Vom Algorithmus aus Abschnitt 4.4.1 ausgehend zeigt sich, daß sich die N-Punkte-DCT auch aus einer N-Punkte-DFT einer reellen Sequenz statt aus einer 2N-Punkte-DFT gewinnen läßt, was zu einer Ersparnis von ca. 50% des Rechenaufwands führt. Dieser Algorithmus wird deshalb auch als "Fast Cosine Transform" (FCT) bezeichnet [4.28].

$$\left.\begin{aligned} \text{Sei}\quad v(n) &= y(2n) \\ u(n) &= y(2n+1) \end{aligned}\right\} \qquad 0 \leq n \leq N-1, \qquad (4.96)$$

wobei $v(n)$ und $u(n)$ die Mengen aller gerade bzw. ungerade indizierten Punkte in $y(n)$ sind. Bild 4.26(b) zeigt, wie diese Teilung im obigen Beispiel vorgenommen wird. $v(n)$ und $u(n)$ enthalten jeweils alle ursprünglichen Abtastun-

gen von x(n), d.h. u(n) ist einfach die Umkehrsequenz von v(n). Mit dieser Tatsache folgt aus (4.85) und (4.89)

$$u(n) = v(N-n-1), \qquad 0 \leq n \leq N-1. \tag{4.97}$$

Indem man (4.96) in (4.86) einsetzt, erhält man

$$Y_f(k) = \sum_{n=0}^{N-1} v(n) W_{2N}^{2nk} + \sum_{n=0}^{N-1} u(n) W_{2N}^{(2n+1)k}. \tag{4.98}$$

Nun substituiert man (4.97) in (4.98) und ordnet die Terme unter Beachtung von $W_{2N}^{2nk} = W_N^{nk}$ um. Dann wendet man (4.91) an und erhält

$$X_c(k) = 2\,\mathrm{Re}\{W_{4N}^{k} \sum_{n=0}^{N-1} v(n)\, W_N^{nk}\}, \qquad 0 \leq k \leq N-1. \tag{4.99}$$

Der Unterschied zwischen (4.99) und (4.92) liegt in der Verwendung von W_N statt W_{2N} in der Summation sowie in der Verwendung von v(n) statt x(n). Es bestätigt sich also, daß man die DCT aus einer N-Punkte-DFT statt 2N-Punkte-DFT berechnen kann. Gleichung (4.99) läßt sich umschreiben als

$$X_c(k) = 2 \sum_{n=0}^{N-1} v(n)\, \cos((4n+1)k\pi/(2N)), \qquad 0 \leq k \leq N-1, \tag{4.100}$$

was eine alternative Definition der DCT (ausgedrückt durch die umgeordnete Sequenz v(n)) ergibt (vgl. (4.100) mit (4.90)).

Die Sequenz v(n) kann direkt durch x(n) dargestellt werden:

$$v(n) = \begin{cases} x(2n), & 0 \leq n \leq \lceil (N-1)/2 \rceil \\ x(2N-2n-1), & \lceil (N+1)/2 \rceil \leq n \leq N-1 \end{cases} \tag{4.101}$$

Deshalb wird v(n) dadurch gewonnen, daß die gerade indizierten Punkte in x(n) in der ursprünglichen Ordnung und anschließend die ungerade indizierten Punkte in umgekehrter Reihenfolge benutzt werden.

Man beachte, daß (4.101) auf beliebige Werte von N (gerade oder ungerade) angewandt werden kann. Die Spezifikation der FCT ist damit vollständig. Weil v(n) reell ist, kann ihre N-Punkte-DFT aus der (N/2)-DFT einer komplexen Sequenz berechnet werden (vgl. Abschnitt 4.2.7).

Nun wird die inverse FCT (IFCT) diskutiert. Dabei wird zunächst v(n) aus der DCT berechnet. Dann wird (4.101) zur Erzeugung von x(n) angewandt. Nach der Substitution v(n) = y(2n) in (4.95) hat man

$$v(n) = N^{-1}\ \mathrm{Re}\{Y_f(0)/2 + \sum_{k=1}^{N-1} Y_f(k)W_N^{-nk}\}, \qquad 0 \leqq n \leqq N-1. \tag{4.102}$$

Gleichung (4.102) deutet darauf hin, daß v(n) unter Verwendung einer N-IDFT statt einer durch (4.95) beschriebenen 2N-IDFT berechnet werden kann.

Die Anzahl der Operationen ist jedoch immer noch etwa zweimal so groß wie bei der Vorwärts-FCT. Die IFCT kann jedoch mit der gleichen Anzahl von Operationen wie die Vorwärts-FCT berechnet werden. Die Methode hierzu ist, aus $X_c(k)$ zunächst $V_f(k)$ zu ermitteln und dann die IDFT von $V_f(k)$ zu berechnen, um v(n) zu erzeugen:

Gleichung (4.99) läßt sich umschreiben als

$$X_c(k) = \mathrm{Re}\{2\ W_{4N}^k V_f(k)\}. \tag{4.103}$$

$V_f(k)$ sei die DFT von v(n). Um $V_f(k)$ aus $X_c(k)$ in (4.103) zu berechnen, müssen wir auch den imaginären Teil des Klammerausdrucks beachten. Wenn der imaginäre Teil und der reelle Teil von $X_c(k)$ mit $C_r(k)$ bzw. mit $C_i(k)$ und die gesamte komplexe Zahl mit $C_c(k)$ bezeichnet wird, wobei

$$C_c(k) = C_r(k) + jC_i(k) = 2\ W_{4N}^k V_f(k), \tag{4.104}$$

dann ergibt sich

$$V_f(k) = (1/2)W_{4N}^{-k}\ C_c(k).$$

Zuerst wird $C_i(k)$ bestimmt. Weil $V_f(k)$ hermitisch symmetrisch ist, d.h.

$$V_f(N-k) = V_f^*(k), \tag{4.105}$$

erhält man unter Verwendung von (4.104)

$$C_c(N-k) = -jC_c^*(k) = -\{C_i(k) + jC_r(k)\}. \tag{4.106}$$

Aus (4.104) und (4.106) ermittelt man

$$C_c(k) = C_r(k) - jC_r(N-k) = 2W_{4N}^{k} V_f(k) \tag{4.107}$$

und mit (4.107) ergibt sich

$$V_f(k) = (1/2)W_{4N}^{-k} \{C_r(k) - jC_r(N-k)\}, \qquad 0 \leqq k \leqq N-1. \tag{4.108}$$

$V_f(k)$ ist aus (4.108) für $0 \leqq k \leqq N/2$ unter Verwendung von (4.105) für $k > N/2$. zu berechnen. Zur Berechnung von $V_f(0)$ braucht man den Wert von $X_c(N)$, der jedoch nach (4.100) und (4.94) gleich null ist, d.h. $C_r(N) = 0$.

Nach Berechnung von $V_f(k)$ wird v(n) durch inverse DFT gewonnen

$$v(n) = N^{-1} \sum_{k=0}^{N-1} V_f(k) W_N^{-nk}. \tag{4.109}$$

Es scheint also, daß in (4.109) wieder eine komplexe N-IDFT benötigt wird. Im Abschnitt 4.2.7 wurde jedoch gezeigt, wie die IDFT einer hermitisch symmetrischen Sequenz unter Verwendung einer komplexen (N/2)-Punkte-IDFT berechnet werden kann. Zur Veranschaulichung fassen wir hier das Verfahren für die Berechnung der FCT und der IFCT zusammen.

FCT-Algorithmus:

Sei x(n) eine reelle Sequenz, $0 \leqq n \leqq N-1$ gegeben.

(1) Bilden der Sequenz v(n) aus (4.101).

(2) Berechnen von $V_f(k)$, $0 \leqq k \leqq N-1$, (DFT von v(n)).

(3) Multiplizieren von $V_f(k)$ mit $\exp(-j\pi k/(2N))$. Aus (4.107) sieht man, daß der reelle Teil $C_r(k)$ ist und der negative Imaginärteil $C_r(N-k)$ ist. Deshalb liegen die Werte von k im Bereich $0 \leqq k \leqq \lceil N/2 \rceil$.

IFCT-Algorithmus:

Sei die DCT $X_c(k)$, $0 \leqq k \leqq N-1$ und $X_c(N) = 0$ gegeben.

(1) Berechnen von $V_f(k)$ aus (4.108),

(2) Berechnen der IDFT von $V_f(k) \longleftrightarrow v(n)$,

(3) Ermittlung von x(n) aus v(n) unter Verwendung von (4.101).

Diese FCT- und IFCT-Algorithmen lassen sich sehr leicht implementieren. Außerdem ist das Verfahren durchaus allgemein und kann für beliebige Werte N benutzt werden. Wie wir gesehen haben, können die N-Punkte-DFT von v(n) und die N-Punkte-IDFT von $V_f(k)$ jeweils aus der (N/2)-Punkte-DFT und der (N/2)-Punkte-IDFT der komplexen Sequenzen berechnet werden. Für große Werte von N wird man natürlich die FFT vorteilhaft benutzen. In diesem Fall ist die Gesamtzahl der Operationen für die FCT bzw. die IFCT von der Ordnung $N\log_2 N$, wenn man die Tatsache der reellen Sequenz ausnutzt.

Zur Berechnung von $X_c(k)$ aus (4.107) bestimmt man zuerst die N-Punkte-DFT von v(n). Deshalb braucht man eine Tabelle von Sinus oder Cosinus, wobei der Einheitskreis in N gleiche Segmente geteilt wird. Man benötigt außerdem eine nachträgliche Multiplikation mit W_{4N}^k, wobei der Einheitskreis in 4N gleiche Segmente geteilt wird. Weil k im Bereich $0 \leq k \leq N-1$ ist, liegen die N Werte von Sinus und Cosinus alle im ersten Quadranten. Daraus folgt die Tatsache, daß die DCT einer N-Sequenz eine Exponentialfunktions-Tabelle benötigt, die 4-mal so groß ist, wie für eine N-Punkte DFT.

Wir können auch anders berücksichtigen, daß die Datensequenz x(n) in zwei Teile (gerade und ungerade indizierte) aufgeteilt wird [4.32]. Aus

$$X_c(k) = 2c(k) \sum_{n=0}^{N-1} x(n)\cos((2n+1)k\pi/(2N)), \qquad k = 0,1,\ldots,N-1$$

und

$$c(k) = \begin{cases} 1/\sqrt{2} & \text{für } k=0, \\ 1 & \text{für } k = 1,2,\ldots,N-1 \end{cases} \tag{4.110}$$

folgt mit

$$\left.\begin{aligned} y(n) &= x(2n), \\ y(N-1-n) &= x(2n+1) \end{aligned}\right\} \quad n = 0,1,\ldots,(N/2)-1,$$

der Ausdruck

$$X_c(k) = 2c(k) \sum_{n=0}^{(N/2)-1} y(n)\cos((4n+1)k\pi/(2N)) + \sum_{n=0}^{(N/2)-1} y(N-1-n)\cos((4n+3)k\pi/(2N)).$$

Mit der Substitution n'= N-1-n in der zweiten Summe, kann man die beiden Summen kombinieren und erhält

$$X_c(k) = 2c(k) \sum_{n=0}^{N-1} y(n)\cos((4n+1)k\pi/(2N)).$$

Nun kann man schreiben $X_c(k) = \mathrm{Re}\{T_f(k)\}$, wobei

$$T_f(k) = \exp(jk\pi/(2N)) \sum_{n=0}^{N-1} y(n)\exp(j2kn\pi/N). \qquad (4.111)$$

Wird die IDFT von y(n) mit $Y_i(k)$ bezeichnet, k = 0,1,...,N-1, kann (4.111) folgendermaßen umgeschrieben werden: $T_f(k) = \exp(j\pi/(2N))Y_i(k)$. Wegen der Symmetrie gilt außerdem $T_f(N-k) = jT_f(k)^*$. Aufgrund dieser Eigenschaft kann die Sequenz $X_c(k)$ durch Berechnung von $T_f(k)$ dargestellt werden. Durch Trennung von Real- und Imaginärteil erhält man

$$\left.\begin{aligned} X_c(k) &= \mathrm{Re}\{T_f(k)\} \\ X_c(N-k) &= \mathrm{Im}\{T_f(k)\}. \end{aligned}\right\} \quad k = 0,1,\ldots,(N/2),$$

Die Anzahl der komplexen Multiplikationen, die zur Auswertung von $Y_i(k)$ (N-Punkte-IDFT der reellen Sequenz y(n)) benötigt wird, ergibt sich bei Verwendung des Radix-2-FFT-Algorithmus' zu $M_c = \{(N/2)\log_2 N-(3N/2)+2\}/2$. Dabei ist angenommen, daß zwei reelle Transformationen durch eine komplexe N-Punkte-FFT ausgewertet werden. Außerdem benötigt man (N/2)-1 komplexe Multiplikationen zur Auswertung von $T_f(k)$ aus $Y_i(k)$. Dies führt zu einer Gesamtzahl von $(N\log_2 N-N+2)/4$ komplexen Multiplikationen für die Berechnung der N-Punkte-DCT. Das ist ungefähr halb so viel wie bei der 2N-Punkte-FFT-Methode [4.28]. Wenn man vier reelle Multiplikationen zum komplexen Multiplizieren benötigt, erfordert dieser Algorithmus $N\log_2 N-N+2$ reelle Multiplikationen.

Die inverse DCT kann in ähnlicher Weise berechnet werden. Sie ist als

$$x(n) = N^{-1} \sum_{n=0}^{N-1} c(k)X_c(k)\cos((2n+1)k\pi/(2N)), \qquad n = 0,1,\ldots,N-1$$

definiert, wobei c(k) in (4.110) gegeben wurde.

Für gerade indizierten Datenpunkte erhält man

$$x(2n) = \mathrm{Re}\{\sum_{k=0}^{N-1} [c(k)X_c(k)\exp(jk\pi/(2N))]\exp(2jkn\pi/N)\}, \quad n = 0,1,\ldots,(N/2)-1.$$

Um die ungerade indizierten Datenpunkte zu erzeugen, setzt man

$$x(2n+1) = x(2(N-1-n)).$$

Daher kann die Datensequenz x(n) durch die Berechnung des reellen Teils der IDFT der Sequenz $c(k)X_c(k)\exp(jk\pi/(2N))$ bestimmt werden. Die Anzahl der erforderlichen Operationen für die IDCT ist die gleiche wie für die Vorwärts-DCT.

4.4.3 DCT-Berechnung aus DHYT-Koeffizienten

Die diskrete Hartley-Transformation (DHYT) ist eine interessante Alternative zur diskreten Fourier-Transformation. Da die DHYT wegen ihrer reellen Basisfunktionen nur reelle Operationen verlangt, liegt es nahe, sie zur Berechnung der (reellen) DCT heranzuziehen. In diesem Abschnitt wird eine Methode für die Berechnung der DCT dargestellt, bei der eine DHYT eines permutierten Datensatzes zuerst berechnet wird, woraus sich die DCT durch Koordinaten-Rotation ergibt [4.31].

Wir betrachten die Beziehung zwischen der DCT und der DHYT einer Datensequenz x(n). Wie angegeben, lautet die DCT der Folge x(n)

$$X_c(k) = \sum_{n=0}^{N-1} x(n)\cos((2n+1)k\pi/(2N)). \tag{4.112}$$

Zur Vereinfachung wird in (4.112) der normalisierende Faktor $1/\sqrt{2}$ für $X_c(0)$ nicht mehr in Betracht gezogen. Sodann wird eine neue Sequenz y(n) definiert

$$y(n) = \begin{cases} x(2n) & \text{für } n = 0,1,\ldots,(N/2)-1 \\ x(2N-2n-1) & \text{für } n = (N/2),\ldots,N-1. \end{cases} \tag{4.113}$$

Die DHYT einer Folge x(n) wurde im Abschnitt 2.2 definiert. Damit ist die DHYT der Folge y(n)

$$Y_{HY}(k) = \sum_{n=0}^{N-1} y(n)\,cas(2nk\pi/N). \qquad (4.114)$$

Aus (4.113) und (4.114), erhält man

$$Y_{HY}(k) = \sum_{n:gerade} x(n)cas(kn\pi/N) + \sum_{n:ungerade} x(n)cas(-(n+1)k\pi/N).$$

Unter Verwendung der Beziehung $cas(\alpha)cas(\beta) = cos(\alpha-\beta)+sin(\alpha+\beta)$ folgt hieraus

$$X_c(k) = (1/2)\{Y_{HY}(k)cas(-k\pi/(2N)) + Y_{HY}(N-k)cas(k\pi/(2N))\}.$$

In Matrixform dargestellt hat man

$$\begin{bmatrix} X_c(k) \\ X_c(N-k) \end{bmatrix} = (1/2) \begin{bmatrix} cas(-\pi k/(2N)) & cas(\pi k/(2N)) \\ cas(\pi(N-k)/(2N)) & cas(-\pi(N-k)/(2N)) \end{bmatrix} \begin{bmatrix} Y_{HY}(k) \\ Y_{HY}(N-k) \end{bmatrix}. \qquad (4.115)$$

Die (2x2)-Matrix in (4.115), ist offensichtlich orthogonal, wodurch gute numerische Eigenschaften sichergestellt sind. Nach Berechnung von $Y_{HY}(k)$ durch einen schnellen DHYT-Algorithmus kann man (4.115) für die Berechnung der DCT-Koeffizienten mit den Indizes $k = 1,2,\ldots,(N/2)-1$ benutzen. Die restlichen Koeffizienten sind gegeben durch

$$X_c(0) = Y_{HY}(0) \qquad \text{und} \qquad X_c(N/2) = (1/\sqrt{2})\, Y_{HY}(N/2).$$

Der Algorithmus für die DCT der Länge N benötigt die Berechnung einer DHYT gleicher Länge, (N/2)-1 Transformationen mit der (2x2)-Matrix in (4.115) und eine Multiplikation zur Erzeugung von $X_c(N/2)$. Weil die Multiplikation eines Vektors mit der (2x2)-Matrix durch drei Multiplikationen und drei Additionen durchgeführt werden kann, ergibt sich für die Anzahl der benötigten Additionen (A) und Multiplikationen (M) für N>4:

$$A(DCT) = A(DHYT) + 3N/2 - 3, \quad \text{und} \quad M(DCT) = M(DHYT) + 3N/2 - 2.$$

In [4.31] wird die Berechnungskomplexität dieser Methode mit anderer DCT-Al-

gorithmen verglichen. Es zeigt sich, daß das Verfahren der Berechnung der DCT aus der DHYT etwa die gleiche Komplexität wie andere effektive DCT-Algorithmen besitzt.

4.4.4 DCT-Algorithmus mit Matrixfaktorisierung

In diesem Abschnitt wird ein anderer DCT-Algorithmus diskutiert [2.8], der auf Matrixfaktorisierung beruht und nur reelle Operationen für die Berechnung der DCT einer Folge von N Punkten benötigt. Dieser Algorithmus kann für jeden beliebigen Wert $N=2^r$ $(r>2)$ benutzt werden. Er stellt nicht unbedingt die effizienteste Form der DCT dar, erlaubt aber methodische Erweiterungen.

Die DCT läßt sich in Matrixform als $\underline{X}_c = [C(L)]\,\underline{x}$ schreiben, wobei $\underline{x}$ ein Datenvektor der Ordnung N und $\underline{X}_c$ der Koeffizientenvektor ist $(N=2^r)$. Damit gilt für das allgemeine Element der DCT-Matrix [C(L)]

$$[c_{nk}] = [c(k)\cos((2n+1)k\pi/(2N))],$$

$$n,k = 0,1,\ldots,N-1, \quad N = 2^r \quad \text{und} \quad L = 1,2,\ldots,r.$$

[C(L)] kann rekursiv zerlegt werden

$$[C(L)] = [P(L)]\begin{bmatrix} [C(L-1)] & 0 \\ 0 & [R(L-1)] \end{bmatrix}[B(L)], \tag{4.116}$$

wobei $[R(L-1)] = [c(k)\cos((2n+1)(2k+1)\pi/(2N))], \quad n,k = 0,1,\ldots,(N/2)-1$

und

$$[B(L)] = \begin{bmatrix} [I(L-1)] & [\overline{I}(L-1)] \\ [\overline{I}(L-1)] & -[I(L-1)] \end{bmatrix}, \quad [\overline{B}(L)] = \begin{bmatrix} -[I(L-1)] & [\overline{I}(L-1)] \\ [\overline{I}(L-1)] & [I(L-1)] \end{bmatrix} \tag{4.117}$$

sind. [I(L)] ist die Identitätsmatrix der Ordnung L. $[\overline{I}(L)]$ ist gegenüber der Identitätsmatrix [I(L)] gespiegelt

$$[\overline{I}(L)] = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 0 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ & & & \dots & & & \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ & & & \dots & & & \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 1 & 0 & 0 & \dots & 0 & 0 & 0 \end{bmatrix}.$$

[P(L)] ist eine Permutationsmatrix, die den Vektor $\underline{X}_c(k)$ aus einer bitvertauschten in eine natürliche Ordnung permutiert. Die DCT-Matrix [C(L)] kann rekursiv aus [C(1)] sowie durch Zerlegung von [R(L-1)] verallgemeinert werden. Dabei ist

$$[C(1)] = (1/\sqrt{2}) \begin{bmatrix} 1 & -1 \\ & \\ 1 & -1 \end{bmatrix}$$

und $[R(L-1)] = [M_1(L-1)] \cdot [M_2(L-1)] \dots [M_{2r-3}(L-1)]$,

oder abgekürzt $[R] = [M_1][M_2][M_3][M_{2r-3}]$.

Die Matrizen $[M_i]$, $1 \leq i \leq 2r-3$ sind von regelmäßiger diagonaler Struktur. Für ihre Herleitung wird auf die Originalliteratur verwiesen [2.8]. Nachfolgend wird ihr Aufbau dargestellt. Dabei werden neben (4.117) die folgenden Abkürzungen verwendet

$$[S(k)(i)] = (\sin(k\pi/i))[I_{N/(2i)}], \qquad [SI(k)(i)] = (\sin(k\pi/i))[\overline{I}_{N/(2i)}]$$
$$[C(k)(i)] = (\cos(k\pi/i))[I_{N/(2i)}], \qquad [CI(k)(i)] = (\cos(k\pi/i))[\overline{I}_{N/(2i)}].$$

Man kann vier verschiedene Typen unterscheiden: gerade und ungerade indizierte Matrizen sowie die erste (linke) und die letzte (rechte) Matrix. Die gerade indizierten Matrizen $[M_p]$, $p=2,4,\dots$ sind binäre Diagonalmatrizen, die aus den Untermatrizen $[B_Z]$ aufbaut sind, wobei $Z=2^p$.

Die ungerade indizierten Matrizen $[M_q]$, $q=3,5,\dots$ bestehen aus Untermatrizen wie aus den nachfolgenden Beispielen zu ersehen ist:

$$[M_1] = \begin{bmatrix} [S(a1)(2N)] & & & [CI(a1)(2N)] \\ & [S(a2)(2N)] & & [CI(a2)(2N)] \\ & & \ddots & \\ & & [S(aN/4)(2N)] & [CI(aN/4)(2N)] \\ & & [S(a)(2N)] & [C(aN/4)(2N)] \\ & & \ddots & \\ & -[S(a)(2N)] & & [C(aN/2-1)(2N)] \\ -[S(aN/2)(2N)] & & & [C(aN/2)(2N)] \end{bmatrix},$$

$$[M_2] = [\mathrm{diag}\{[B_2], [\overline{B}_2], [\overline{B}_2], [B_2], \ldots, [\overline{B}_2], [B_2]\}],$$

$$[M_3] = \begin{bmatrix} 1 & & & 0 \\ & -[C(b1)(N/2)] & [SI(b1)(N/2)] & \\ & -[S(b1)(N/2)] & -[CI(b1)(N/2)] & \\ & 1 & 0 & \\ & & \ddots & \\ & 0 & 1 & \\ & -[SI(bN/8)(N/2)] & -[C(bN/8)(N/2)] & \\ & -[SI(bN/8)(N/2)] & [S(bN/8)(N/2)] & \\ 0 & & & 1 \end{bmatrix},$$

$$[M_4] = [\mathrm{diag}\{[B_4], [\overline{B}_4], [B_4], [\overline{B}_4], \ldots, [B_4], [\overline{B}_4]\}],$$

$$[M_5] = \begin{bmatrix} [I(1)] & & & 0 \\ & -[C(c1)(N/4)] & [SI(c1)(N/4)] & \\ & -[S(c1)(N/4)] & -[CI(c1)(N/4)] & \\ & [I(1)] & 0 & \\ & & \ddots & \\ & 0 & [I(1)] & \\ & -[SI(cN/16)(N/4)] & -[C(cN/16)(N/4)] & \\ & -[CI(cN/16)(N/4)] & [S(cN/16)(N/4)] & \\ 0 & & & [I(1)] \end{bmatrix},$$

$$[M_6] = [\mathrm{diag}\{[B_6], [\overline{B}_6], [B_6], [\overline{B}_6], \ldots, [B_6], [\overline{B}_6]\}].$$

$$[M_{2r-5}] = \begin{bmatrix} [I(r-4)] & & & & & & 0 \\ & -[C_{(1)}(8)] & & & & [SI_{(1)}(8)] & \\ & & -[S_{(1)}(8)] & & -[CI_{(1)}(8)] & & \\ & & & [I(r-4)] & & 0 & \\ & & & & \ddots & & \\ & & & 0 & & [I(r-4)] & \\ & & -[SI_{(3)}(8)] & & & -[C_{(3)}(8)] & \\ & -[CI_{(3)}(8)] & & & & & [S_{(3)}(8)] \\ 0 & & & & & & [I(r-4)] \end{bmatrix}.$$

$$[M_{2r-4}] = [\mathrm{diag}\{[B_{N/4}], [\overline{B}_{N/4}]\}].$$

$$[M_{2r-3}] = \begin{bmatrix} [I(r-3)] & & 0 \\ & -[C_{(1)}(4)] \quad [CI_{(1)}(4)] & \\ & & \\ & [CI_{(1)}(4)] \quad [C_{(1)}(4)] & \\ 0 & & [I(r-3)] \end{bmatrix}.$$

Wir betrachten nun ein spezielles Beispiel für r=4 (N=16)

$$[R(3)] = [M_1] \cdot [M_2]\ [M_3] \cdot [M_4]\ [M_5].$$

Dabei sind die Faktoren:

$$[M_1(3)] = \begin{bmatrix} \sin(\pi/32) & & & & \cos(\pi/32) \\ & \sin(9\pi/32) & & \cos(9\pi/32) & \\ & & \sin(5\pi/32) & & \cos(5\pi/32) \\ & & \sin(13\pi/32) & \cos(13\pi/32) & \\ & & & & \\ & & -\sin(3\pi/32) & \cos(3\pi/32) & \\ & -\sin(11\pi/32) & & & \cos(11\pi/32) \\ & & -\sin(7\pi/32) & & \cos(7\pi/32) \\ -\sin(15\pi/32) & & & & \cos(15\pi/32) \end{bmatrix}.$$

$$[M_2(3)] = \begin{bmatrix} 1 & 1 & & & & & & \\ 1 & -1 & & & & & & \\ & & -1 & 1 & & & & \\ & & 1 & 1 & & & & \\ & & & & 1 & 1 & & \\ & & & & 1 & -1 & & \\ & & & & & & -1 & 1 \\ & & & & & & 1 & 1 \end{bmatrix},$$

$$[M_3(3)] = \begin{bmatrix} 1 & & & & & & & 0 \\ -\cos(\pi/8) & & & & & & \sin(\pi/8) & \\ & -\sin(\pi/8) & & & & -\cos(\pi/8) & & \\ & & & 1 & 0 & & & \\ & & & & & & & \\ & & & 0 & 1 & & & \\ & -\sin(3\pi/8) & & & & \cos(3\pi/8) & & \\ -\sin(3\pi/8) & & & & & & \cos(3\pi/8) & \\ 0 & & & & & & & 1 \end{bmatrix},$$

$$[M_4(3)] = \begin{bmatrix} 1 & & & 1 & & & & \\ & 1 & 1 & & & & & \\ & 1 & -1 & & & & & \\ 1 & & & -1 & & & & \\ & & & & -1 & & & 1 \\ & & & & & -1 & 1 & \\ & & & & & 1 & 1 & \\ & & & & 1 & & & 1 \end{bmatrix}$$

und

$$[R_5(3)] = \begin{bmatrix} 1 & & & & & & & 0 \\ & 1 & & & & & 0 & \\ & & -\cos(\pi/4) & & & \cos(\pi/4) & & \\ & & & -\cos(\pi/4) & \cos(\pi/4) & & & \\ & & & & & & & \\ & & & \cos(\pi/4) & \cos(\pi/4) & & & \\ & & \cos(\pi/4) & & & \cos(\pi/4) & & \\ & 0 & & & & & 1 & \\ 0 & & & & & & & 1 \end{bmatrix}.$$

Unter Beachtung von (4.116) läßt sich nun der SFG konstruieren. Im Bild 4.27 ist dieser für den Fall N=16 dargestellt.

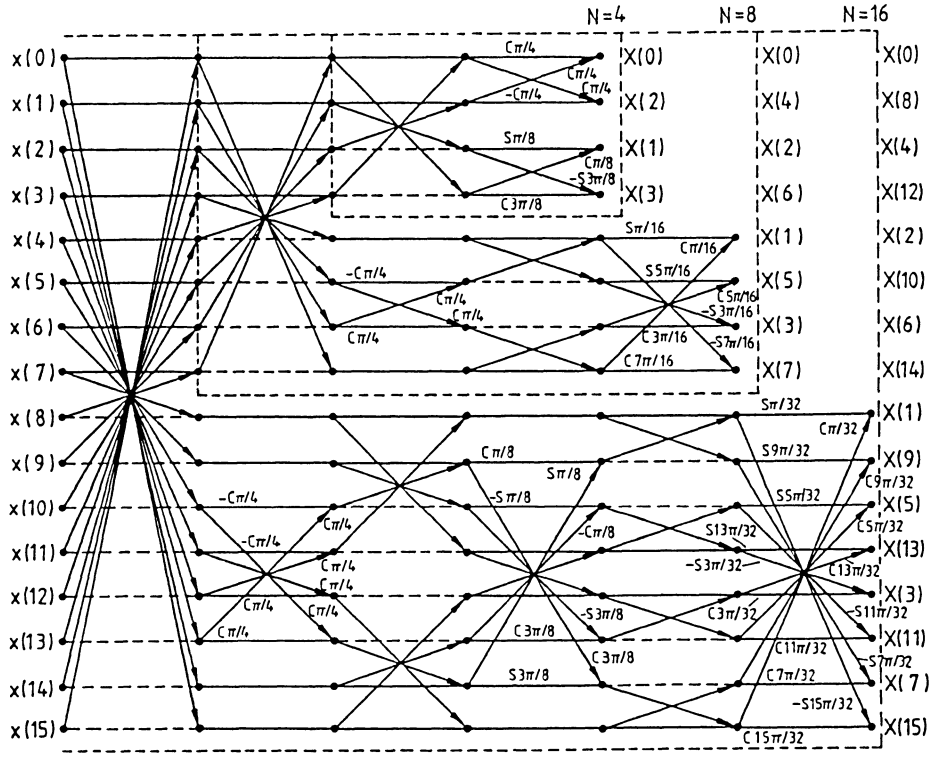


Bild 4.27. Signalflußgraph zur effizienten Berechnung der DCT für N=4, 8 und 16. (Zur Bezeichnungsvereinfachung sind die Multiplikatoren mit $C\vartheta$ und $S\vartheta$ statt $\cos(\vartheta)$ und $\sin(\vartheta)$ abgekürzt, nach [2.8])

Zu diesem DCT-Algorithmus und seinen SFG sind noch folgende Anmerkungen von Interesse:

a) Der Signalflußgraph für den Fall $N=2^r$ enthält automatisch die Signalflußgraphen für $N=2^{r-1}$, $N=2^{r-2}$,..., $N=2^2$.

b) Die DCT-Koeffizienten erscheinen in Bitumkehrordnung, während die Objektdaten in natürlicher Ordnung sind.

c) Für jedes N werden die gerade indizierten Koeffizienten direkt aus den Koeffizienten der (N/2)-Punkte-DCT durch Verdoppelung der Indexziffern erhalten.

d) Zur Erweiterung des SFG auf den nächsthöheren Wert von r ist links eine Spalte von binären Schmetterlingen hinzuzufügen. Die ungerade indizierten Koeffizienten werden durch Spalten von abwechselnd Cosinus/Sinus-Schmetterlingen und binären Schmetterlingen erzeugt (vgl. Bild 4.27).

e) Die DCT-Koeffizienten gemäß Bild 4.27 sind mit dem Multiplikator N/2 zu normalisieren.

f) Da $[C(L)]$ eine orthogonale Matrix ist, ergibt sich die inverse DCT einfach als $[C(L)]^{-1} = [C(L)]^T$.

g) Dieser Algorithmus für die N-Punkte-FCT benötigt

$(3N/2)((\log_2 N)-1) + 2$ reelle Additionen

und

$N(\log_2 N) - (3N/2) + 4$ reelle Multiplikationen.

Das sind ungefähr 1/6 der Operationen die zur DCT-Berechnung durch eine 2N-Punkte-FFT [4.28] erforderlich sind (vgl. Abschnitt 4.4.1). Bei Yip und Rao findet man ein Rechenverfahren ähnlicher Art für die DST [4.32] (vgl. Abschnitt 4.5.1).

4.4.5 Algorithmus zur direkten DCT-Berechnung

Ein anderer Algorithmus, der direkt von der DCT-Definition ausgeht und deshalb nur reelle Arithmetik verwendet, wurde von B.G. Lee [4.34] angegeben. Im Gegensatz zu dem im Abschnitt 4.4.3 beschriebenen Verfahren geht der Algorithmus von Lee von der Summenform aus, die in gerade und ungerade indizierte Anteile zerlegt wird.

Die Summenform der inversen DCT (IDCT) lautet

$$x(n) = \sum_{k=0}^{N-1} c(k)\, X_c(k)\cos((2n+1)k\pi/(2N)), \qquad n = 0,1,\ldots,N-1,$$

wobei c(k) durch (4.110) definiert ist.

Zur Vereinfachung der Schreibweise wird

$$C_{2N}^{(2n+1)k} = \cos((2n+1)k\pi/(2N))$$

gesetzt. Dann stellt sich die N-Punkte-IDCT als

$$x(n) = \sum_{k=0}^{N-1} \hat{X}(k) C_{2N}^{(2n+1)k} \tag{4.118}$$

dar, wobei $\hat{X}(k) = c(k)X_c(k)$.

Durch Zerlegung von x(n) in Folgen mit geraden und ungeraden Indizes k erhält man aus (4.118)

$$x(n) = g(n) + h(n), \qquad x(N-1-n) = g(n) - h(n), \tag{4.119}$$

wobei

$$g(n) = \sum_{k=0}^{(N/2)-1} \hat{X}(2k)\, C_{2N}^{(2n+1)2k}, \tag{4.120}$$

$$h(n) = \sum_{k=0}^{(N/2)-1} \hat{X}(2k+1)\, C_{2N}^{(2n+1)(2k+1)}, \qquad n = 0,1,\ldots,(N/2)-1. \tag{4.121}$$

g(n) stellt eine (N/2)-Punkte-IDCT dar, weil

$$C_{2N}^{(2n+1)2k} = C_{N}^{(2n+1)k} = C_{2(N/2)}^{(2n+1)k}.$$

Nun wird h(n) in folgender Form geschrieben

$$h(n) = \sum_{k=0}^{(N/2)-1} \tilde{X}(2k+1)\, C_{2(N/2)}^{(2n+1)(k+(1/2))}, \tag{4.122}$$

was wiederum eine (N/2)-Punkte-IDCT ist. Weil

$$2C_{2N}^{(2n+1)}\, C_{2N}^{(2n+1)(2k+1)} = C_{2N}^{(2n+1)2k} + C_{2N}^{(2n+1)2(k+1)}, \tag{4.123}$$

folgt aus (4.122) und (4.123)

$$2C_{2N}^{(2k+1)}h(n) = \sum_{k=0}^{(N/2)-1} \tilde{X}(2k+1)\, C_{2N}^{(2n+1)(2k)} + \tilde{X}(2k+1)\, C_{2N}^{(2n+1)2(k+1)}. \tag{4.124}$$

Mit der Definition $\tilde{X}(2k-1) = 0$ für $k = 0$ folgt unter Verwendung von

$$C_{2N}^{(2n+1)2(N/2)} = C_{2}^{(2n+1)} = 0$$

$$\sum_{k=0}^{(N/2)-1} \tilde{X}(2k+1)C_{2N}^{(2n+1)2(k+1)} = \sum_{k=0}^{(N/2)-1} \tilde{X}(2k-1)C_{2N}^{(2n+1)2k}.$$

Deshalb kann (4.124) wie folgt umgeschrieben werden

$$2C_{2N}^{(2n+1)}h(n) = \sum_{k=0}^{(N/2)-1} \{\tilde{X}(2k+1) + \tilde{X}(2k-1)\}C_{2N}^{(2n+1)2k}. \tag{4.125}$$

Weiter werden folgende Definitionen benutzt:

$$g(n) = \sum_{k=0}^{(N/2)-1} G(k)\, C_{2(N/2)}^{(2n+1)k} \qquad G(k) = \tilde{X}(2k)$$

$$h(n) = \sum_{k=0}^{(N/2)-1} H(k)\, C_{2(N/2)}^{(2n+1)k} \qquad H(k) = \tilde{X}(2k+1) + \tilde{X}(2k-1),$$

$$n = 0,1,\ldots,(N/2)-1, \qquad k = 0,1,\ldots,(N/2)-1. \tag{4.126}$$

Dann erhält man mit (4.119), (4.120), (4.121) und (4.125) bis (4.126) das endgültige Ergebnis:

$$x(n) = g(n) + \{1/(C_{2N}^{(2n+1)})\}h(n),$$

$$x(N-1-n) = g(n) - \{1/(2C_{2N}^{(2n+1)})\}h(n), \qquad n = 0,1,\ldots,(N/2)-1. \tag{4.127}$$

So ist die N-Punkte-IDCT in die Summe von zwei (N/2)-Punkte-DCT (4.126) zerlegt. Durch Wiederholung dieses Prozesses kann die IDCT weiter zerlegt werden.

In ähnlicher Weise kann man auch die DCT-Matrix zerlegen. Weil die DCT eine orthogonale Transformation ist, kann die DCT-Matrix auch durch Transponieren der IDCT-Matrix bzw. durch Umdrehen der Richtungen der Pfeile im Flußgraph der IDCT gewonnen werden.

Nun betrachten wir ein Beispiel mit N=8: Mit (4.124) bis (4.127) erhält man

$$G(k) = \tilde{X}(2k), \qquad H(k) = \tilde{X}(2k+1) + \tilde{X}(2k-1), \qquad k = 0,1,2,3 \tag{4.128}$$

und

$$g(n) = \sum_{k=0}^{3} G(k)\, C_8^{(2n+1)k}, \tag{4.129}$$

$$h(n) = \sum_{k=0}^{3} H(k)\, C_8^{(2n+1)k}, \tag{4.130}$$

sowie

$$x(n) = g(n) + \{1/(2C_{16}^{(2n+1)})\}h(n), \qquad n = 0,1,2,3, \tag{4.131}$$

$$x(7-n) = g(n) - \{1/(2C_{16}^{(2n+1)})\}h(n), \qquad n = 0,1,2,3. \tag{4.132}$$

Die Gleichungen (4.128), (4.131) und (4.132) korrespondieren jeweils mit der ersten und der letzten Iteration im Flußgraph vom Bild 4.28. Durch Wiederholung dieser Schritte für die Gleichungen (4.129) und (4.130) erhalten wir den FCT-Flußgraph für eine 8-Punkte-IDCT wie im Bild 4.28 dargestellt.

Die Anzahl der reellen Multiplikationen für eine N-Punkte-FCT mit $N=2^r$ beträgt $(N/2)(\log_2 N)-N+1$. Die Anzahl der Additionen ist geringfügig höher, nämlich $(3N/2)(\log_2 N)-N+1$. Die Tabelle 4.9 zeigt einen Vergleich der Anzahl der Operationen mit dem im Abschnitt 4.4.4 beschriebenen Algorithmus.

Der Preis für die geringere Anzahl von Operationen besteht in einer sehr speziellen Ordnung der Ein- und Ausgangsdaten. Die Eingangssequenz $\tilde{X}(k)$ ist in Bitumkehrordnung. Die Anordnung der Ausgangssequenz x(n) wird in folgender Weise erzeugt. Angefangen mit der Indexmenge (0,1) wird eine neue Menge

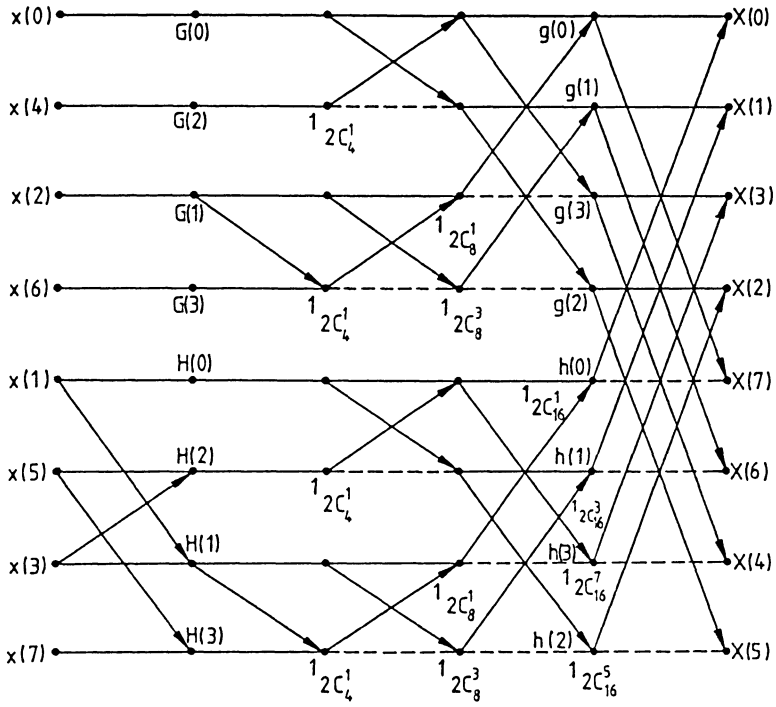


Bild 4.28. DCT-Flußgraph für eine 8-Punkte-IDCT, nach [4.34]

Tabelle 4.9. Vergleich der Algorithmen nach Abschnitt 4.4.5 und Abschnitt 4.4.4, [4.34]

r	N	Anzahl der Multiplikationen		Anzahl der Additionen	
		Chen u.a.	Lee	Chen u.a.	Lee
3	8	16	12	26	29
4	16	44	32	74	81
5	32	116	80	194	209
6	64	292	192	482	513
7	128	708	448	1154	1217
8	256	1668	1024	2690	2817
9	512	3844	2304	6146	6401
10	1024	8708	5120	13826	14337

durch Zufügen einer führenden 0 zu jedem Element erzeugt. Die übrigen Elemente werden durch Negation der Vorhandenen generiert. Dieser Prozeß führt zur Menge (00, 01, 11, 10). Durch Wiederholung erhalten wir dann (000, 001, 011, 010, 111, 110, 100, 101). Für den Fall N=8 folgt damit die Ausgangssequenz x(0), x(1), x(3), x(2), x(7), x(6), x(4) und x(5).

4.4.6 DCT-Berechnung aus Koeffizienten von Teilfolgen

Wir betrachten einen Algorithmus für die DCT, der die Cosinus-Transformationskoeffizienten höherer Ordnung aus solchen niedrigerer Ordnung erzeugt [3.1].

Aus dem Walsh-Hadamard-Koeffizientenvektor $\underline{X}_{WH}(k)$ kann der Cosinus-Koeffizientenvektor $\underline{X}_c(k)$ erzeugt werden (vgl. Abschnitt 3.1.3)

$$\underline{X}_c(<k>) = [T_k(L)]\,\underline{X}_{WH}(<k>), \qquad L = 1,2,\ldots,r, \quad r = \log_2 N, \tag{4.133}$$

wobei <k> die Bitumkehrordnung von k bedeutet, und $[T_k(L)]$ eine N×N-Transformationsmatrix ist. Umgekehrt ist

$$\underline{X}_{WH}(k) = [T_k(L)]^{-1}\underline{X}_c(k) = [T_k(L)]^T\underline{X}_c(k).$$

Die Matrix $[T_k(L)]$ ist orthogonal und hat eine blockdiagonale Struktur der Form

$$[T_k(L)] = \begin{bmatrix} [T_k(L-1)] & 0 \\ \hline 0 & [R_k(L-1)] \end{bmatrix}, \tag{4.134}$$

wobei $[T_k(L-1)]$ eine Konversionsmatrix der Ordnung (N/2)×(N/2) ist. Eine Restmatrix der gleichen Ordnung ist $[R_k(L-1)]$.

Der Datenvektor mit $N=2^r$ Elementen sei mit $\underline{x}$ bezeichnet. Schreibt man die Vektoren seiner DCT-Koeffizienten als $\underline{X}_c$ und seiner WHT-Koeffizienten als $\underline{X}_{WH}$, dann haben wir

$$\underline{X}_c = [C(L)]\,\underline{x}, \qquad \underline{X}_{WH} = [H_{CS}(L)]\,\underline{x} \tag{4.135}$$

und

$$\underline{x} = N^{-1}[H_{CS}(L)]^T \underline{X}_{WH} = (2\sqrt{2}/N)\ [C(L)]^T \underline{X}_c . \qquad (4.136)$$

wobei [C(L)] und $[H_{CS}(L)]$ die (N×N)-Transformationsmatrizen der Cosinus-Transformation und der $(WHT)_{CS}$ sind.

Aus (4.133), (4.135) und (4.136) folgen die Beziehungen

$$\underline{X}_c = (1/\sqrt{8})[C(L)][H_{CS}(L)]^T \underline{X}_{WH}$$

und $$[T_k(L)] = (1/\sqrt{8})[C_{<k>}(L)][H_{CS<k>}(L)]^T . \qquad (4.137)$$

Dabei sind [C(L)] und $[H_{CS}(L)]$ orthonormale Matrizen. Deshalb ist ihr Produkt $[T_k(L)]$ auch eine orthonormale Matrix. Hadamard-Matrizen höherer Ordnung können aus Matrizen niedrigerer Ordnung erzeugt werden. Außerdem kann $[T_k(L)]$ iterativ beschrieben werden, wie die Gleichung (4.134) zeigt.

Wie schon im Abschnitt 4.1.2 benutzt, wird der WHT-Koeffizientenvektor in zwei Hälften aufgeteilt

$$\underline{X}_{WH}(L) = \begin{bmatrix} \underline{X}_{WH1}(L-1) + \underline{X}_{WH2}(L-1) \\ - - - - - - - - - - - - - - - \\ \underline{X}_{WH1}(L-1) - \underline{X}_{WH2}(L-1) \end{bmatrix} . \qquad (4.138)$$

$\underline{X}_{WH1}(L-1)$ und $\underline{X}_{WH2}(L-1)$ beziehen sich je auf die WHT der oberen und der unteren Hälfte des Datenvektors $\underline{x}$.

Substitution von (4.138) in (4.133) und Anwendung von (4.134) führen zu einer rekursiven Beziehung für $\underline{X}_c$

$$\underline{X}_c(L) = \begin{bmatrix} [T_k(L-1)] & 0 \\ - - - - - & - - - - - \\ 0 & [R_k(L-1)] \end{bmatrix} \begin{bmatrix} \underline{X}_{WH1}(L-1) + \underline{X}_{WH2}(L-1) \\ - - - - - - - - - - - - - - - \\ \underline{X}_{WH1}(L-1) - \underline{X}_{WH2}(L-1) \end{bmatrix} . \qquad (4.139)$$

Unter Verwendung von (4.133) und (4.139) und Bezeichnung der DCT-Vektoren mit $\underline{X}_{c1}(L-1)$ und $\underline{X}_{c2}(L-1)$ ergibt sich

$$\underline{X}_c(L) = \begin{bmatrix} [T_k(L-1)] & 0 \\ - - - - - & - - - - - \\ 0 & [R_k(L-1)] \end{bmatrix} \begin{bmatrix} [T_k(L-1)]^T \underline{X}_{c1}(L-1) + [T_k(L-1)]^T \underline{X}_{c2}(L-1) \\ - \\ [T_k(L-1)]^T \underline{X}_{c1}(L-1) - [T_k(L-1)]^T \underline{X}_{c2}(L-1) \end{bmatrix} .$$

Weil $[T_k(L-1)]$ eine unitäre Matrix ist, erhält man schließlich

$$\underline{X}_c(L) = \begin{bmatrix} \underline{X}_{c1}(L-1) + \underline{X}_{c2}(L-1) \\ - \\ [R_k(L-1)][T_k(L-1)]^T\{\underline{X}_{c1}(L-1) - \underline{X}_{c2}(L-1)\} \end{bmatrix}. \tag{4.140}$$

Gleichung (4.140) zeigt, daß die DCT-Koeffizienten N-ter Ordnung aus den DCT-Koeffizienten der oberen Hälfte des Vektors $\underline{x}$ und aus den DCT-Koeffizienten der unteren Hälfte des Vektors $\underline{x}$ erzeugt werden können.

Zur Vereinfachung der Schreibweise setzt man

$$\underline{X}_c(L) = \begin{bmatrix} \underline{X}_{c1}(L-1) + \underline{X}_{c2}(L-1) \\ - - - - - - - - - - - - - - - - - \\ [Z_k(L-1)](\underline{X}_{c1}(L-1) - \underline{X}_{c2}(L-1)) \end{bmatrix},$$

wobei $[Z_k(L-1)] = [R_k(L-1)][T_k(L-1)]^T$. (4.141)

Es folgt ein Beispiel für die Erzeugung der DCT-Koeffizienten eines Vektors mit N=8 Elementen aus den Cosinus-Transformationskoeffizienten von zwei Teilvektoren der Ordnung N=4.

Benutzen wir beispielsweise (4.141) für L=3

$$\underline{X}_c(3) = \begin{bmatrix} \underline{X}_{c1}(2) + \underline{X}_{c2}(2) \\ - - - - - - - - - - - - - - - \\ [Z_k(2)](\underline{X}_{c1}(2) - \underline{X}_{c2}(2)) \end{bmatrix}. \tag{4.142}$$

$[T_k(3)]$, $[R_k(2)]$ und $[T_k(2)]$ werden dadurch berechnet, daß die Transformationsmatrizen $[H_{CS}(3)]$ und $[C(3)]$ angewandt werden. Mit (4.137) erhält man daraus die Matrix $[T_k(3)]$. Aus ihr folgen $[T_k(2)]$ und $[R_k(2)]$:

$$[H_{CS}(3)] = \left[\begin{array}{cccc|cccc|ll} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 000 & 000 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 001 & 100 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 010 & 010 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 011 & 110 \\ & & & & & & & & & \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 100 & 001 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 101 & 101 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 110 & 011 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 111 & 111 \end{array}\right]$$

Natürl. Index — BRO-Index

$$[C(3)] = \left[\begin{array}{cccc|cccc} 0,354 & 0,354 & 0,354 & 0,354 & 0,354 & 0,354 & 0,354 & 0,354 \\ 0,490 & 0,416 & 0,278 & 0,098 & -0,098 & -0,278 & -0,416 & -0,490 \\ 0,462 & 0,191 & -0,191 & 0,462 & -0,462 & -0,462 & -0,191 & 0,462 \\ 0,416 & -0,098 & -0,490 & -0,278 & 0,278 & 0,490 & 0,098 & -0,416 \\ & & & & & & & \\ 0,354 & -0,354 & -0,354 & 0,354 & 0,354 & -0,354 & -0,354 & 0,354 \\ 0,278 & -0,490 & 0,098 & 0,416 & -0,416 & -0,098 & 0,490 & -0,278 \\ 0,191 & -0,462 & 0,462 & -0,191 & -0,191 & 0,462 & -0,462 & 0,191 \\ 0,098 & -0,278 & 0,416 & -0,490 & 0,490 & -0,416 & 0,278 & -0,098 \end{array}\right].$$

$$[T_k(3)] = (1/\sqrt{8})[C_{<k>}(3)][H_{<k>}(3)] = (1/\sqrt{8})\left[\begin{array}{c|c} [T_k(2)] & 0 \\ \hline 0 & [R_k(2)] \end{array}\right].$$

$$[T_k(3)] = \left[\begin{array}{cccccccc} 1,0 & & & & & & & \\ & 1,0 & & & & & & \\ & & 0,923 & 0,383 & & & 0 & \\ & & -0,383 & 0,923 & & & & \\ & & & & & & & \\ & & & & 0,907 & -0,075 & 0,375 & 0,180 \\ & & 0 & & 0,214 & 0,768 & -0,513 & 0,318 \\ & & & & -0,318 & 0,513 & 0,768 & 0,214 \\ & & & & -0,180 & -0,375 & -0,075 & 0,907 \end{array}\right].$$

Matrix $[Z_k(2)]$ kann nun aus (4.141) berechnet und abgespeichert werden. Sind nun zwei DCT-Koeffizientenvektoren der Ordnung 4 gegeben, können die DCT-Koeffizienten des zusammengesetzten Vektors der Ordnung 8 aus (4.142) berechnet werden.

Dieser Algorithmus ist zweckmäßig, wenn nacheinander die Koeffizienten von Datensequenzen jeweils doppelter Länge berechnet werden sollen. Normalerweise müßten jeweils die Daten im Speicher gehalten oder bei "in-place"-Verarbeitung durch inverse DCT rückgewonnen werden.

Zur Abschätzung der Effizienz wird die Anzahl der in diesem Verfahren benötigten Operationen mit der des "in-place"-Verfahrens verglichen. Bei dieser Methode wird eine inverse DCT der Ordnung (N/2) zur Erzeugung der Datenvektoren im Objektbereich aus den DCT-Koeffizienten der Größe (N/2) benutzt. Dann wird eine DCT der Ordnung N zur Berechnung der DCT-Koeffizienten der Ordnung N verwendet. Es zeigt sich, daß die hier beschriebene Methode für Vektoren der Ordnung N<128 den geringeren Rechenaufwand erfordert [3.1].

4.4.7 Algorithmus für eine modififizierte DCT

Die günstigen Eigenschaften der DCT haben sie in der Signalverarbeitung zu einem wichtigen Werkzeug gemacht. Insbesondere bei der Codierung von Sprach- und Bildsignalen spielt sie eine herausragende Rolle, da ihre dekorrelierenden Eigenschaften für gewisse Modellsignale an die der optimalen KLT (Karhunen-Loève-Transformation) heranreichen. Obwohl die DCT im Gegensatz zur KLT einen "schnellen" Algorithmus besitzt, und im Gegensatz zur DFT keine komplexen Operationen benötigt, hat sie gegenüber der WHT den Nachteil, daß zahlreiche Multiplikationen mit Cosinus-Funktionswerten erforderlich sind.

Es ist deshalb versucht worden, vereinfachte Formen zu entwickeln, die mit weniger Rechenaufwand ähnlich gute Eigenschaften erreichen. Eine solche Transformation ist die modifizierte DCT (MDCT) [4.35], die wir in diesem Abschnitt beschreiben. Zuerst wird die Gerade/Ungerade-Transformation vorgestellt und anschließend zur Walsh-Hadamard- und zur Cosinus-Transformation benutzt. Der Vorteil der einfachen Berechnung der WHT wird hier zur Gewinnung der MDCT-Koeffizienten ausgenutzt.

Die Transformationsmatrix [T(L)] der Ordnung NxN einer Gerade/Ungerade-Transformation $(EOT)_N$ kann durch eine orthogonale und blockdiagonale Struktur in der folgenden Form beschrieben werden

$$[T(L)] = \begin{bmatrix} [T(L-1)] & [\overline{T}(L-1)] \\ [R(L-1)] & -[\overline{R}(L-1)] \end{bmatrix}$$

$$= \begin{bmatrix} [T(L-1)] & 0 \\ 0 & -[\overline{R}(L-1)] \end{bmatrix} \begin{bmatrix} [I(L-1)] & [\overline{I}(L-1)] \\ [\overline{I}(L-1)] & -[I(L-1)] \end{bmatrix}. \qquad (4.143)$$

wobei [T(L-1)] eine (N/2)x(N/2)-Konversionsmatrix, [R(L-1)] eine Restmatrix und [I(L-1)] die Identitätsmatrix der Ordnung (N/2)x(N/2) ist. Die überstrichenen Matrizen sind gespiegelte Matrizen.

Die Transformationsmatrix [T(L-1)] kann wiederum rekursiv dargestellt werden

$$[T(L-1)] = \begin{bmatrix} [T(L-2)] & [\overline{T}(L-2)] \\ [R(L-2)] & -[\overline{R}(L-2)] \end{bmatrix}. \qquad (4.144)$$

Die Betrachtung von (4.143) und (4.144) führt zu folgender Beschreibeung:

$$[T(L)] = \begin{bmatrix} [T(L-2)] & 0 & 0 \\ 0 & [\overline{R}(L-2)] & \\ & & \\ 0 & & [\overline{R}(L-1)] \end{bmatrix}$$

$$\times \begin{bmatrix} [I(L-2)]\cdot[\overline{I}(L-2)] & 0 \\ -[\overline{I}(L-2)]\cdot[I(L-2)] & \\ & \\ 0 & [I(L-1)] \end{bmatrix} \begin{bmatrix} [I(L-1)] & [\overline{I}(L-1)] \\ [\overline{I}(L-1)] & -[I(L-1)] \end{bmatrix}.$$

Die Gerade/Ungerade-Struktur der Transformation wird zuerst für die $(WHT)_w$-Matrix in Bitumkehrordnung dargestellt:

$$[H_w(3)] = [T(3)] = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ \hline 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{array}\right]$$

$$= \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & & & & \\ 1 & -1 & -1 & 1 & & 0 & & \\ 1 & 1 & -1 & -1 & & & & \\ 1 & -1 & 1 & -1 & & & & \\ \hline & & & & -1 & -1 & -1 & -1 \\ & & & & -1 & 1 & 1 & -1 \\ & 0 & & & 1 & 1 & -1 & -1 \\ & & & & 1 & -1 & 1 & -1 \end{array}\right] \left[\begin{array}{cccc|cccc} 1 & & & & & & & 1 \\ & 1 & & & & & 1 & \\ & & 1 & & & 1 & & \\ & & & 1 & 1 & & & \\ \hline & & & 1 & 1 & & & \\ & & 1 & & & 1 & & \\ & 1 & & & & & 1 & \\ 1 & & & & & & & 1 \end{array}\right].$$

$$[T(2)] = \left[\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ & & & \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{array}\right] = \left[\begin{array}{cccc} 1 & 1 & & 0 \\ 1 & 1 & & \\ & & & \\ & & 1 & 1 \\ 0 & & 1 & 1 \end{array}\right] \left[\begin{array}{cccc} 1 & & & 1 \\ & 1 & 1 & \\ & & & \\ & 1 & 1 & \\ 1 & & & 1 \end{array}\right].$$

$$[R(2)] = \left[\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ & & & \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{array}\right] = \left[\begin{array}{cccc} 1 & 1 & & 0 \\ 1 & 1 & & \\ & & & \\ & & 1 & 1 \\ 0 & & 1 & 1 \end{array}\right] \left[\begin{array}{cccc} 1 & & & 1 \\ & 1 & 1 & \\ & & & \\ & 1 & 1 & \\ 1 & & & 1 \end{array}\right].$$

Nun wird der MDCT-Algorithmus diskutiert, der unter Verwendung der EOT die Berechnung der MDCT einer Folge von N Punkten ermöglicht. Dieser Algorithmus kann auf jeden beliebigen Wert $N=2^r$ erweitert werden.

Wir betrachten die Matrixschreibweise für die DCT

$$\underline{X}_c = [C(L)]\ \underline{x},$$

wobei $\underline{x}$ und $\underline{X}_c$ der Datenvektor bzw. der Koeffizientenvektor sind. Die Transformationsmatrix hat folgenden Elemente

$$[c_{nk}] = [c(k)\cos((2n+1)k\pi/(2N))], \qquad n,\ k = 0,1,\ldots,N-1$$

mit

$$c(k) = \begin{cases} 1/\sqrt{2} & \text{für } k=0 \\ 1 & \text{für } k = 1,2,\ldots,N-1. \end{cases}$$

[C(L)] kann unter Verwendung der EOT in schwach besetzte Matrizen zerlegt werden:

$$[R(L)] = \begin{bmatrix} [C(L-1)] & 0 \\ \hline 0 & [R(L-1)] \end{bmatrix} \begin{bmatrix} [I(L-1)] & [\overline{I}(L-1)] \\ \hline [\overline{I}(L-1)] & -[I(L-1)] \end{bmatrix},$$

$$L = 1,2,\ldots,r, \qquad r = \log_2 N.$$

Die Transformationsmatrix [C(L)] kann rekursiv aus [R(L-1)] und durch Zerlegung von [C(L-1)] verallgemeinert werden:

$$[R(1)] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

$$[C(L)] = \begin{bmatrix} [S_r(L-1)] & 0 \\ \hline 0 & [S_r(L-1)] \end{bmatrix} [U(L)],$$

$$[S_r(L)] = \begin{bmatrix} [S_r(L-1)] & 0 \\ 0 & [S_r(L-1)] \end{bmatrix} \begin{bmatrix} [I(L-1)] & [\overline{I}(L-1)] \\ [\overline{I}(L-1)] & -[I(L-1)] \end{bmatrix},$$

mit

$$[S_r(1)] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Dabei werden die Matrizen [U(L)] wie folgt entwickelt:

$$[U(1)] = \begin{bmatrix} \sin(\pi/8) & \cos(\pi/8) \\ \cos(\pi/8) & -\sin(3\pi/8) \end{bmatrix},$$

$$[U(2)] = \begin{bmatrix} \sin(\pi/16) & \cos(\pi/16) \\ \sin(3\pi/16) & \cos(3\pi/16) \\ \cos(3\pi/16) & -\sin(3\pi/16) \\ \cos(\pi/16) & -\sin(\pi/16) \end{bmatrix},$$

$$[M_1(3)] = \begin{bmatrix} \sin(\pi/32) & \cos(\pi/32) \\ \sin(3\pi/32) & \cos(3\pi/32) \\ \sin(5\pi/32) & \cos(5\pi/32) \\ \sin(7\pi/32) & \cos(7\pi/32) \\ \cos(7\pi/32) & -\sin(7\pi/32) \\ \cos(5\pi/32) & -\sin(5\pi/32) \\ \cos(3\pi/32) & -\sin(3\pi/32) \\ \cos(\pi/32) & -\sin(\pi/32) \end{bmatrix}.$$

Die so entstandene SFG-Struktur wird als modifizierte DCT (MDCT) bezeichnet [4.35].

Der SFG der MDCT ist im Bild 4.29 für N=8 dargestellt. Die Konstanten a_0, a_1, a_2, b_0, b_1, und b_2 werden so gewählt, daß einige der Basisvektoren gleich denen der DCT werden. Zum Beispiel:

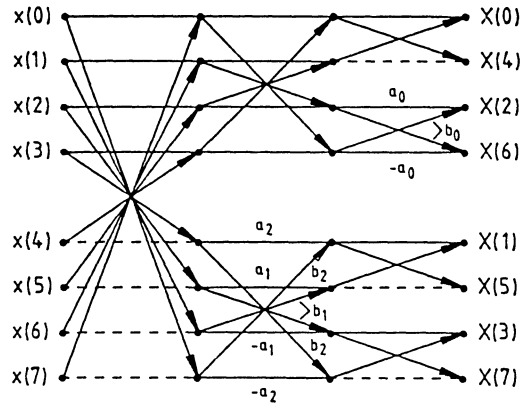


Bild 4.29. SFG der MDCT für N=8, nach [4.35]

$a_0 = \sin(\pi/8)$, $a_1 = \sin(3\pi/16)$, $a_2 = \sin(\pi/16)$,
$b_0 = \cos(\pi/8)$, $b_1 = \cos(3\pi/16)$, $b_2 = \cos(\pi/16)$.

Die Transformation entspricht nicht genau der DCT. Nach Untersuchung von Kanchan und Rao [4.35] liegen ihre Eigenschaften, z.B bezüglich der Dekorrelation ihrer Koeffizienten, zwischen denen der "wahren" DCT und der Walsh-Transformation. Dafür benötigt man bei der MDCT den gleichen Aufwand an Additionen wie für die WHT und zusätzlich nur 2N-4 Multiplikationen.

Eine ausführliche Darstellung der diskreten Cosinus-Transformation und ihrer Anwendungen findet man in [4.33].

4.5 Algorithmen für die diskrete Sinus-Transformation

4.5.1 DST-Algorithmus mit Matrixfaktorisierung

Die Algorithmen für die DST verwenden i. allg. den traditionellen Ansatz, d.h. es wird die 2N-Punkte-FFT einer Sequenz von N Punkten zur Berechnung der DST benutzt. Der imaginäre Teil wird dann herausgezogen, um die N-Punkte-DST anzugeben.

Yip und Rao [4.32] haben einen schnellen Algorithmus für die diskrete Sinus-Transformation der Definition I (DST(I)) entwickelt (vgl. Abschnitt 2.3),

ähnlich dem DCT-Algorithmus von Chen [2.8]. Im folgenden betrachten wir diesen DST-Algorithmus.

Die Matrixfaktorisierung ist von der gleichen Art wie sie für die DCT im Abschnitt 4.4.4 beschrieben wurde. Die rekursive Struktur ist daher sehr ähnlich derjenigen für die DCT.

Im Abschnitt 2.3 wurde die DST(I) $X_s(k)$ der Datensequenz x(n) und ihre Inverse wie folgt definiert:

$$X_s(k) = \sqrt{2/(N+1)} \sum_{n=1}^{N} x(n)\sin(nk\pi/(N+1)),$$

$$x(n) = \sqrt{2/(N+1)} \sum_{k=1}^{N} X_s(k)\sin(nk\pi/(N+1)).$$

Sei $N = 2^r-1$ und [S(N)] die DST-Matrix, dann gilt

$$[S_N] = \sqrt{2/(N+1)}\,[P_N][A_N].$$

$[P_N]$ ist eine Zeilenpermutationsmatrix, die durch Bitumkehr von (n-1) definiert wird (n ist der Zeilenindex, d.h. n = 1,2,...,N). Das allgemeine Element von $[S_N]$ in der n-ten Zeile und der k-ten Spalte Φ_{nk} ist durch

$$\Phi_{nk} = \sqrt{2/(N+1)}\,\sin(nk\pi/(N+1))$$

beschrieben.

$[A_N]$ besitzt alle Strukturen, die für eine Faktorisierung notwendig sind. Der erste Schritt liegt in der rekursiven Beziehung

$$[A_N] = \begin{bmatrix} [A_{(N+1)/2}] & 0 \\ 0 & [A_{(N-1)/2}] \end{bmatrix} [B_N]. \qquad (4.145)$$

Dabei ist $[B_N]$ eine Schmetterlingsmatrix, die nur entlang der beiden Diagonalen von Null verschiedene Elemente des Betrags 1 hat, wobei die untere Hälfte der Hauptdiagonale mit negativen Elementen besetzt ist, z.B.

$$[B_5] = \begin{bmatrix} 1 & & & & 1 \\ & 1 & & 1 & \\ & & 1 & & \\ & 1 & & -1 & \\ 1 & & & & -1 \end{bmatrix}.$$

Die Inverse von $[B_5]$ hat die gleiche Struktur, außer einem Element 2 in der Mitte und einem normalisierenden Faktor 1/2:

$$[B_5]^{-1} = (1/2)\begin{bmatrix} 1 & & & & 1 \\ & 1 & & 1 & \\ & & 2 & & \\ & 1 & & -1 & \\ 1 & & & & -1 \end{bmatrix}.$$

Die wesentliche rekursive Struktur liegt im unteren Diagonalblock der Größe ((N–1)/2)×((N–1)/2) des ersten Faktors von (4.145).

Eine weitere Faktorisierung von $[A_{(N+1)/2}]$ ist eine [(N+1)/2]×[(N+1)/2]–Matrix, die genauer betrachtet werden soll. Die niedrigste Ordnung der nichttrivialen Transformationsmatrix $[A_3]$ wird angegeben durch

$$[A_3] = \begin{bmatrix} [A_2] & 0 \\ 0 & 1 \end{bmatrix}[B_3].$$

Dabei ist

$$[A_2] = \begin{bmatrix} \sin(\pi/4) & \sin(\pi/2) \\ \sin(\pi/4) & -\sin(\pi/2) \end{bmatrix} = 1/\sqrt{2}\begin{bmatrix} 1 & \sqrt{2} \\ 1 & -\sqrt{2} \end{bmatrix}.$$

Die nächsthöhere Ordnung ist N=7. Für sie gilt

$$[A_7] = \begin{bmatrix} [A_4] & 0 \\ 0 & [A_3] \end{bmatrix}[B_7].$$

Die Matrix im oberen Diagonalblock von (4.145) ist

$$[A_{(N+1)/2}] = [P_{(N+1)/2}]\ [\sin((2n+1)(k+1)\pi/(N+1))], \qquad n,k = 0,1,\ldots,(N-1)/2.$$

Dies kann weiter in

$$[A_{(N+1)/2}] = [B_{(N+1)/2}]\ [D_{(N+1)/2}]\ [J_{(N+1)/2}]$$

faktorisiert werden.

Dabei hat $[B_{(N+1)/2}]$ die gleiche Schmetterlingsstruktur wie $[B_5]^{-1}$, abgesehen vom in der Mitte liegenden Element 2, denn $(N+1)/2 = 2^{r-1}$.

$[J_{(N+1)/2}]$ ist eine Spaltenpermutationsmatrix, die alle ungeraden Spalten nach links und alle geraden Spalten nach rechts sortiert, d.h.

$$[J_{(N+1)/2}] = \begin{bmatrix} 1 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 \\ \ldots & & \ldots & & \ldots & \\ & & \ldots & & & \\ \ldots & & \ldots & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & 0 & 1 \end{bmatrix}.$$

Wie man leicht überprüfen kann, ist $[J_{(N+1)/2}][J_{(N+1)/2}] = [I_{(N+1)/2}]$.

Die Matrix $[D_{(N+1)/2}]$ ist wieder diagonal:

$$[D_{(N+1)/2}] = \begin{bmatrix} [F_{(N+1)/4}] & 0 \\ & \\ 0 & [G_{(N-1)/4}] \end{bmatrix}.$$

Hier liegt eine weitere rekursive Beziehung vor. Die Matrix $[G_{(N+1)/4}]$ kann nämlich durch einfache Zeilenoperationen aus $[A_{(N+1)/2}]$ erhalten werden

$$[G_{(N+1)/4}] = [F_{(N+1)/4}][A_{(N+1)/4}].$$

Für die Elemente $[f_{kn}]$ der Matrix $[F_{(N+1)/4}]$ gilt:

$$[f_{kn}] = \begin{cases} -1 & \text{für a=0 mit ungeradem k, n und mit k, n > (N+1)/8,} \\ 1 & \text{für b=1 mit geradem k und geradem n und mit k, n} \leqq \text{(N+1)/8,} \\ 0 & \text{sonst} \end{cases}$$

mit a = ((k+1)/2)−n mod (N+1)/8 bzw. b = ((k/2)+n) mod (N+1)/8.

Zum Beispiel folgt für den Fall N=15

$$[G_4] = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ & & & \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{bmatrix} [A_4].$$

Die Faktorisierung des oberen Blocks in $[D_{(N+1)/2}]$ gestaltet sich ähnlich wie im Abschnitt 4.4.4 für die DCT, d.h.

$$[F_{(N+1)/4}] = [R^T_{(N+1)/4}][M_{(N+1)/4}(1)] \ \ldots \ [M_{(N+1)/4}(L)] \ \ldots \ [M_{(N+1)/4}(r)].$$

$[R_{(N+1)/4}]$ ist eine Zeilenpermutationsmatrix, die die ungeraden Zeilen in zunehmender Ordnung in die obere Hälfte und die geraden Zeilen in umkehrter Ordnung in die untere Hälfte sortiert. Die Anzahl von Matrixfaktoren $[M_i]$ ist $2\log_2((N+1)/2)-3$ für N>7 [4.32]. Diese Methode ist gleichermaßen für die DST(II) anwendbar, wenn die Transformationslänge $N=2^r-1$ ist. Der Algorithmus benötigt lediglich reelle Arithmetik.

4.5.2 DST-Berechnung über die DCT

Eine Methode zur Berechnung der DST aus der DCT ist Gegenstand dieses Abschnitts. Es ergibt sich, daß die schnelle Sinus-Transformation durch die Implementierung der FCT erhalten werden kann [4.12].

Im Abschnitt 2.3 haben wir die folgende Definition der DCT angegeben

$$X_c(k) = 2c(k) \sum_{n=0}^{N-1} x(n)\cos((2n+1)k\pi/(2N)), \qquad k = 0,1,\ldots,N-1. \qquad (4.146)$$

$$\text{Dabei ist } c(k) = \begin{cases} 1/\sqrt{2} & \text{für } k=0 \\ 1 & \text{für } k = 1,2,\ldots,N-1. \end{cases}$$

Durch Substitution von $k = N - m$ und $m = 1,2,\ldots,N$ in (4.146) haben wir

$$X_c(N-m) = 2c(N-m) \sum_{n=0}^{N-1} x(n)\cos((2n+1)(N-m)\pi/(2N)).$$

Weil

$$\cos((2n+1)(N-m)\pi/(2N)) = \cos((2n+1)\pi/2)\cos((2n+1)m\pi/(2N))$$

$$+ \sin((2n+1)\pi/2)\sin((2n+1)m\pi/(2N))$$

$$= (-1)^n \sin((2n+1)m\pi/(2N)),$$

folgt

$$X_c(N-m) = 2c(N-m) \sum_{n=0}^{N-1} x(n)(-1)^n \sin((2n+1)m\pi/(2N)).$$

Sei $\overline{x}(n) = (-1)^n x(n)$ und $\overline{X}_c(k)$ die DCT der Folge $\overline{x}(n)$. Dann ergibt sich

$$X_c(N-m) = 2c(N-m) \sum_{n=0}^{N-1} \overline{x}(n) \sin((2n+1)m\pi/(2N))$$

$$= 2c(N-m) \sum_{n=0}^{N-1} x(n) \sin((2n+1)m\pi/(2N)) = X_s(m).$$

Dabei ist $X_s(m)$ die diskrete Sinus-Transformierte der Definition II (DST(II)) der Datensequenz $\overline{x}(n)$. Deshalb besteht das Verfahren der Gewinnung der DST der Folge x(n) aus folgenden drei Schritten:

(1) Umkehren der Vorzeichen aller ungerade indizierten Daten zur Erzeugung einer neuen Sequenz $\overline{x}(n)$.

(2) Berechnen der DCT der Sequenz $\overline{x}(n)$. Die von Chen [2.8] entwickelte FCT ist eine geeignete Implementierung dieses Schrittes.

(3) Man erhält die DST der Sequenz x(n) durch Umkehrung der Ordnung der in Schritt 2 erzeugten Koeffizientenfolge.

Dieses Verfahren kann als Matrixmultiplikation dargestellt werden. Seien $\underline{X}_s$ und $\underline{X}_c$ die DST- bzw. DCT-Vektoren der Ordnung N, dann ist

$$\underline{X}_s = [\overline{I}_N]\, \underline{X}_c [D_N].$$

Dabei ist $[D_N]$ eine Vorzeichenumkehrmatrix, die wie folgt definiert ist

$$[D_N] = \begin{bmatrix} 1 & & & & & & & 0 \\ & -1 & & & & & & \\ & & 1 & & & & & \\ & & & -1 & & & & \\ & & & & \cdot & & & \\ & & & & & \cdot & & \\ & & & & & & \cdot & \\ & & & & & & 1 & \\ 0 & & & & & & & -1 \end{bmatrix},$$

und $[\overline{I}_N]$ die gespiegelte Einheitsmatrix der Ordnung N. Der Signalflußgraph für die DST(II) entspricht dem im Abschnitt 4.4.4 für DCT angegebenen, an den zwei einfache Operationen angeschlossen werden.

4.6 Algorithmen für die diskrete Haar-Transformation

Für die Durchführung der diskreten Haar-Transformation (DHT) stehen drei Arten von Algorithmen zur Verfügung:

(1) Algorithmus ohne "in-place"-Eigenschaft,
(2) "in-place"-Algorithmus mit Eingangsfolge in natürlicher Ordnung,
(3) "in-place"-Algorithmus mit Eingangsfolge in bitumgekehrter Ordnung.

Wie im Abschnitt 2.5 gezeigt wurde, können die diskrete Haar-Transformation (DHT) und ihre Inverse (IDHT) wie folgt dargestellt werden:

$$X_{HA}(k) = \sum_{n=0}^{N-1} x(n)\ HAR(n,k/N), \qquad k = 0,1,\ldots,N-1, \quad (4.147)$$

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X_{HA}(k)\ HAR(n,k/N), \qquad n = 0,1,\ldots,N-1.$$

Die entsprechende Matrixform wurde ebenfalls im Abschnitt 2.5 angegeben.

Wenn die durch (4.147) definierte DHT direkt ausgeführt wird, sind N^2 Additionen für die Berechnung notwendig. Dies kann auf $r \cdot N$ reduziert werden, wobei $N=2^r$, wenn wir nur die von Null verschiedenen Werte berücksichtigen.

4.6.1 HT-Algorithmus ohne "in-place"-Eigenschaft

Eine beträchtliche Verbesserung der Berechnungseffektivität wird erzielt, wenn eine Zerlegung der Transformationsmatrix in Faktoren verwendet werden kann, ähnlich wie bei der FFT und der FWHT. Der Signalflußgraph eines derartigen Algorithmus' ist im Bild 4.30 dargestellt [2.15].

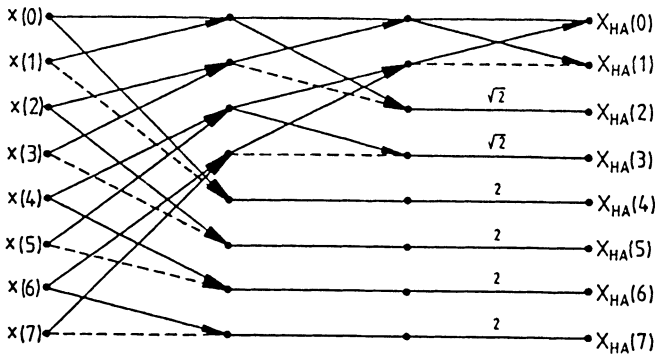


Bild 4.30 SFG der Haar-Transformation (nicht "in-place") für N=8, nach [2.15]

Man beachte die Multiplikationen von Summen oder Differenzen mit $\sqrt{2}$ oder 2 im Bild 4.30. Eine Besonderheit im Vergleich etwa zum Graph der FFT und der FWHT besteht darin, daß dieser Algorithmus nicht "mit Ersetzung" oder speichersparend ("in-place") ausgeführt werden kann. Außer bei der letzten Iteration gelangen die Ergebnisse nicht auf Knoten desselben Niveaus wie die Werte, aus denen sie berechnet wurden.

In der ersten Iteration des Algorithmus' werden N Operationen durchgeführt, in jeder folgenden Iteration ist die Anzahl der Operationen nur halb so groß

wie bei der vorhergehenden Iteration. Die von der Transformation benötigte Anzahl von Additionen bzw. Subtraktionen ist deshalb

$$N + (N/2) + (N/4) + \ldots + 2 = 2(N-1).$$

Diese Anzahl ist geringer als bei der FWHT bzw. der FFT. Außerdem werden 3N/4 Multiplikationen mit Konstanten benötigt. Die Rechenzeit für diesen Algorithmus ist deshalb proportional zu N, im Gegensatz zur FWHT, bei der sie zu $N\log_2 N$ proportional ist.

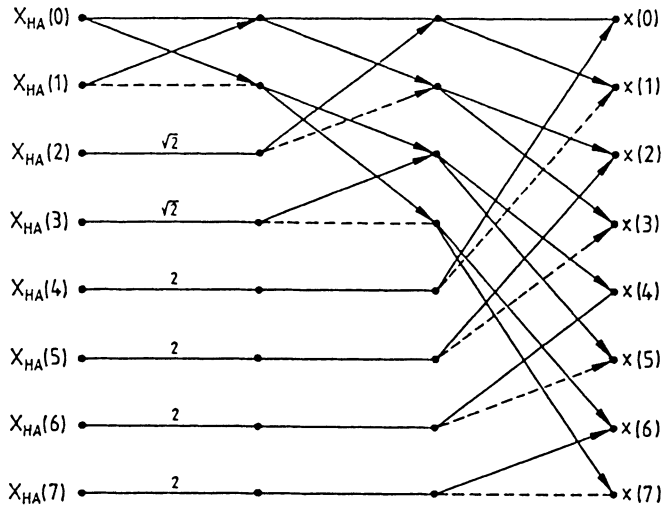


Bild 4.31. SFG der inversen Haar-Transformation (invers zu Bild 4.30), nach [2.15]

Der Signalflußgraph für die inverse Transformation ist im Bild 4.31 dargestellt. Dieser Algorithmus erzeugt geordnete Koeffizienten, braucht jedoch in jeder Iteration zusätzliche Speicherplätze für die Zwischenspeicherung von Werten, die in dieser Iteration berechnet werden. Der Flußgraph führt bei diser FHT-Version mit geordneten Koeffizienten zu einem Aufblähung des Speicherbedarfs. Weil die wesentlichen Operationen der Haar-Transformation einfache Additionen und Subtraktionen sind, wird die Speicherung von Zwischenergebnissen in jeder Iteration eine Ursache unverhältnismäßig langer Rechenzeiten.

4.6.2 Algorithmen zur "in-place"-Berechnung der HT

Im diesem Abschnitt wird der speichersparende ("in-place") Algorithmus diskutiert, der nur die Speicherplätze für die ursprünglichen Funktionswerte benötigt. Diese Variante reduziert außerdem die Anzahl der Operationen. Die Kosten dafür liegen in einer anderen Ordnung der Haar-Koeffizienten, die ggf. der Umsortierung bedarf. Der hohe Speicherbedarf der Transformation nach Abschnitt 4.6.1 und die Notwendigkeit von Umspeicherungen war die Motivation für die Entwicklung dieses Algorithmus' [4.36].

Bei diesem Algorithmus ist in jeder Iteration die Anzahl der Operationen, die zwischengespeichert werden müssen, genau zwei (Radix-2). Wenn jeweils ein derartiges Operationenpaar berechnet ist, können die beiden Speicherplätze zum Abspeichern der Ergebnisse benutzt werden.

Die Notwendigkeit der Datenumspeicherung entfällt. Eine Umordnung der Koeffizienten kommt nach dem Transformationsvorgang infrage, wenn die Koeffizienten in der gewöhnlichen Ordnung von aufsteigenden Indexnummern der Koeffizienten innerhalb der Stufen abgespeichert werden sollen. Um sie zu ordnen, brauchen wir eine zonale Bitumkehroperation. Eine Zone ist eine Menge von Koeffizienten mit Indizes zwischen zwei aufeinanderfolgenden Potenzen von 2. Eine zonale Bitumkehroperation ist eine normale Bitumkehrordnung der Sequenz-Indizes gefolgt von einer Umordnung in die ursprüngliche Ordnung innerhalb jeder Zone [2.15]. Für 8 Koeffizienten ergibt sich die zonale Bitumkehroperation gemäß Tabelle 4.10. Die nach Zonen geordnete Haar-Matrix der Ordnung 8 wurde bereits im Abschnitt 2.5 angegeben:

$$
[\mathrm{HAR}(3)] = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
\sqrt{2}\,(1 & 1 & -1 & -1 & 0 & 0 & 0 & 0) \\
\sqrt{2}\,(0 & 0 & 0 & 0 & 1 & 1 & -1 & -1) \\
 & & & & & & & \\
2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 & -2
\end{bmatrix}
\begin{matrix}
\text{Zonen} \\
0 \\
1 \\
2 \\
2 \\
. \\
3 \\
3 \\
3 \\
3
\end{matrix}
$$

Tabelle 4.10. Zonale Bitumkehroperation für N=8 [3.7]

Index	Binär-Darstellung	Bit-Umkehr	Umordnung innerhalb der Zonen	Neuer Index
0	000	000	000	0
1	001	100	100	4
2	010	010	010	2
3	011	110	110	6
4	100	001	001	1
5	101	101	011	3
6	110	011	101	5

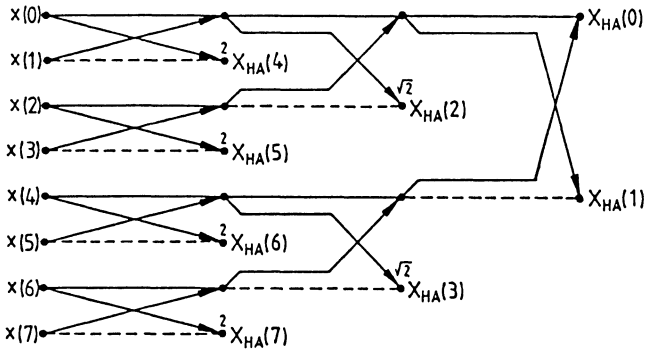


Bild 4.32. SFG der "in-place"-Haar-Transformation für N=8, nach [4.36]

Bild 4.32 illustriert einen Flußgraph wie er sich durch diese Veränderung im Algorithmus ergibt.

Dieser Transformationsalgorithmus hat einen zusätzlichen Vorteil. Die geordneten Koeffizienten benötigen wegen der asymmetrischen Natur des Flußgraphen eine besondere Routine zur inversen Transformation. Der "in-place"-Algorithmus benötigt für die inverse Transformation lediglich andere Werte für die multiplikativen Konstanten (im SFG nicht eingetragen). Ansonsten kann das gleiche Programm sowohl für die Vorwärtstransformation wie für die inverse Transformation benutzt werden.

Eine ähnlicher Algorithmus, bei dem die Datenwerte in bitumgekehrter Ordnung angeordnet sind, wurde von Ahmed und Rao [2.7] angegeben.

4.7 Algorithmen für die Slant-Transformation

4.7.1 SLT-Algorithmus mit Faktorisierung der Transformationsmatrix

Die Slant-Transformation (SLT) wurde im Abschnitt 2.6 definiert. In diesem Abschnitt diskutieren wir Algorithmen für die Slant-Transformation.

Die Slant-Matrix [SL(L)] läßt sich in ein Produkt einer Folge von schwach besetzten Matrizen zerlegen. Das führt zu einem schnellen Algorithmus für die Slant-Transformation. Zum Beispiel gilt für [SL(2)] folgende Matrixzerlegung [2.15]:

$$[SL(2)] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3/\sqrt{5} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3/\sqrt{5} \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/3 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1/3 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix}.$$

Bild 4.33 zeigt den Signalflußgraph der Slant-Transformation mit N=8. Man beachte, daß der Algorithmus nicht r Iterationen sondern (r+1) Iterationen benötigt. Die Slant-Transformation mit N=4 erfordert 8 Additionen und 6 Multiplikationen, während die schnelle Walsh-Transformation nur 8 Additionen benötigt.

4.7.2 SLT-Algorithmus unter Verwendung der Walsh-Transformation

In diesem Abschnitt wird ein Algorithmus für die Slant-Transformation angegeben, der sich aus der im Abschnitt 2.6 erwähnten Methode der Konstruktion von Slant-Matrizen ergibt [4.37].

Wir betrachten zunächst ein Beispiel mit N=8. Aus Abschnitt 2.6 haben wir für die Slant-Matrix mit r=3

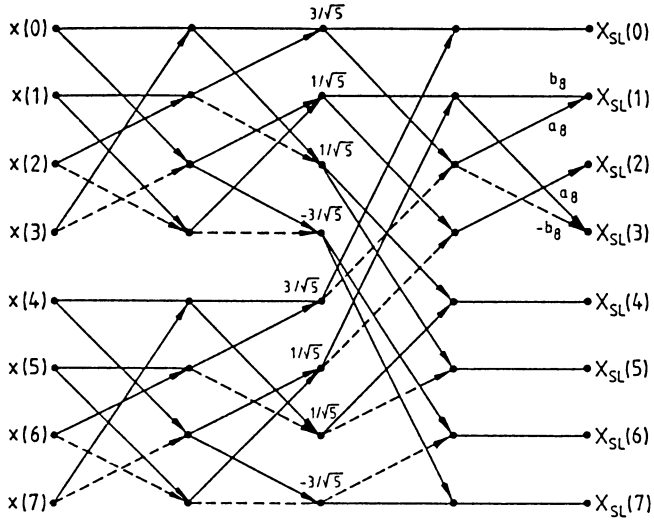


Bild 4.33. SFG der Slant-Transformation für N=8, nach [2.15]

$$[SL'(3)] = (1/\sqrt{2}) \begin{bmatrix} [I_1] & & & \\ \hline & a(3) & -b(3) & \\ & 1 & & \\ & . & & \\ & & 1 & \\ \hline & b(3) & a(3) & \\ & & 1 & \\ & & . & \\ & & & 1 \\ \hline & & & [I_1] \end{bmatrix} \begin{bmatrix} [SL'(2)] & [SL'(2)] \\ \hline [SL'(2)] & -[SL'(2)] \end{bmatrix}. \qquad (4.148)$$

Dabei ist [SL'(3)] die Slant-Matrix der Ordnung $N=2^3$. Der Strich deutet an, daß die Sequenzen in Hadamard-Ordnung vorliegen. Für die Bestimmung der Werte a(r) und b(r) (oder a_N und b_N) siehe Abschnitt 2.6.

Die rechte Matrix in (4.148) kann wie folgt zerlegt werden:

$$\begin{bmatrix} [SL'(2)] & [SL'(2)] \\ \hline [SL'(2)] & -[SL'(2)] \end{bmatrix} = \begin{bmatrix} [I_2] & [I_2] \\ \hline [I_2] & -[I_2] \end{bmatrix} \begin{bmatrix} [SL'(2)] & \\ \hline & [SL'(2)] \end{bmatrix}.$$

Wiederum die rechte Matrix kann weiter zerlegt werden:

$$\begin{bmatrix} [SL'(2)] & \\ \hline & [SL'(2)] \end{bmatrix} = (1/\sqrt{2}) \begin{bmatrix} 1 & & & \\ a(2) & -b(2) & & \\ \hline b(2) & a(2) & & \\ & & 1 & \\ \hline & & & \begin{matrix} 1 & \\ a(2) & -b(2) \\ \hline b(2) & a(2) \end{matrix} \end{bmatrix}$$

$$\times \begin{bmatrix} [SL'(1)] & [SL'(1)] & & \\ [SL'(1)] & -[SL'(1)] & & \\ \hline & & [SL'(1)] & [SL'(1)] \\ & & [SL'(1)] & -[SL'(1)] \end{bmatrix}. \qquad (4.149)$$

Die erste Matrix auf der rechten Seite von (4.149) kann als

$$\begin{bmatrix} [M'(2)] & \\ \hline & [M'(2)] \end{bmatrix} \quad \text{mit} \quad [M'(2)] = \begin{bmatrix} [SL'(1)] & [SL'(1)] \\ [SL'(1)] & -[SL'(1)] \end{bmatrix}$$

gedeutet werden. Weil die Matrizen $[H_p(1)]$ und $[H_p(2)]$ durch

$$[H_p(1)] = [SL(1)] = (1/\sqrt{2}) \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

und

$$[H_p(2)] = (1/\sqrt{2}) \begin{bmatrix} [H_p(1)] & [H_p(1)] \\ \hline [H_p(1)] & -[H_p(1)] \end{bmatrix} = (1/\sqrt{2}) \begin{bmatrix} [SL(1)] & [SL(1)] \\ \hline [SL(1)] & -[SL(1)] \end{bmatrix}$$

darstellbar sind, hat man

$$\begin{bmatrix} [SL'(2)] & 0 \\ \hline 0 & [SL'(2)] \end{bmatrix} = \begin{bmatrix} [M'(2)] & 0 \\ \hline 0 & [M'(2)] \end{bmatrix} \begin{bmatrix} [H_p(2)] & 0 \\ \hline 0 & [H_p(2)] \end{bmatrix}. \qquad (4.150)$$

Nach der Substitution von (4.150) in (4.149) erhält man

$$SL'(3)] = \sqrt{2}\,[M'(3)] \begin{bmatrix} [I(2)] & [I(2)] \\ [I(2)] & -[I(2)] \end{bmatrix} \begin{bmatrix} [M'(2)] & \\ & [M'(2)] \end{bmatrix} \begin{bmatrix} [H_p(2)] & \\ & [H_p(2)] \end{bmatrix}.$$

Da die Multiplikation mit der Einheitsmatrix kommutativ ist, gilt

$$\begin{bmatrix} [M'(2)] & \\ & [M'(2)] \end{bmatrix} \begin{bmatrix} [I(2)] & [I(2)] \\ [I(2)] & -[I(2)] \end{bmatrix} = \begin{bmatrix} [I(2)] & [I(2)] \\ [I(2)] & -[I(2)] \end{bmatrix} \begin{bmatrix} [M'(2)] & \\ & [M'(2)] \end{bmatrix}$$

und

$$\begin{bmatrix} [I(2)] & [I(2)] \\ [I(2)] & -[I(2)] \end{bmatrix} \begin{bmatrix} [H_p(2)] & \\ & [H_p(2)] \end{bmatrix} = \begin{bmatrix} [H_p(2)] & [H_p(2)] \\ [H_p(2)] & -[H_p(2)] \end{bmatrix} = \sqrt{2}\,[H_p(3)].$$

Dabei ist $[H_p(3)]$ die dyadisch geordnete Hadamard-Matrix mit N=8. Daher kann $[SL'(3)]$ unter Verwendung der Hadamard-Matrix erzeugt werden

$$[SL(3)] = [M(3)] \begin{bmatrix} [M'(2)] & \\ & [M'(2)] \end{bmatrix} [H_p(3)]. \qquad (4.151)$$

Ein der Gleichung (4.151) entsprechender Signalflußgraph wird durch Bild 4.34 illustriert.

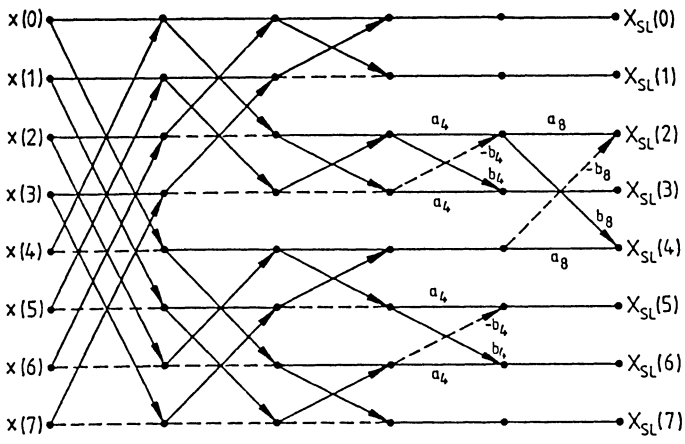


Bild 4.34. SFG der Hadamard-Slant-Transformation für N=8, nach [2.15]

Dieser Algorithmus erfordert noch eine Iteration mehr als der vorhergehende Algorithmus. Allerdings sind die beiden letzten Iterationen sehr schwach "besetzt".

Die ersten drei Iterationen sind identisch mit den entsprechenden der dyadisch geordneten Walsh-Hadamard-Transformation $(WHT)_p$, die nur $N\log_2 N$ Additionen erfordert. Die letzten beiden Iterationen konvertieren die Walsh-Koeffizienten zu Slant-Koeffizienten. Hierzu werden noch (N-1) Additionen und 2(N-2) Multiplikationen benötigt. Deshalb werden für diesen Algorithmus insgesamt $N\log_2 N + N - 2$ Additionen und 2(N-2) Multiplikationen zur Berechnung der Slant-Koeffizienten aus einem Datenvektor der Ordnung N benötigt.

Der SFG für die inverse Slant-Transformation hat eine ähnliche Struktur. Allerdings durchlaufen die Daten den SFG (Bild 4.34) von links nach rechts. In den beiden "konvertierenden" Iterationen sind außerdem die Vorzeichen der Multiplikatoren $b(2) = b_4$ und $b(3) = b_8$ zu vertauschen [4.37].

Eine andere Version des Algorithmus' (Bild 4.35) benutzt die sequenzgeordneten Walsh-Funktionen, deren Koeffizienten in ähnlicher Weise in Slant-Koeffizienten konvertiert werden [2.15].

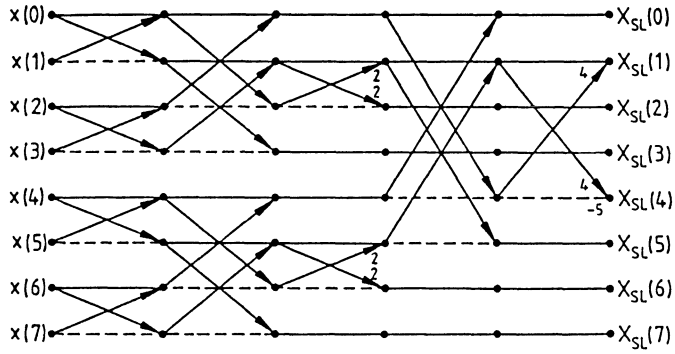


Bild 4.35. SFG der Walsh-Slant-Transformation für N=8, nach [2.15]

4.7.3 SLT-Algorithmus unter Verwendnung der Haar-Transformation

Die im Abschnitt 4.7.2 dargestellten Algorithmen konvertieren die einfach zu erzeugenden Walsh-Koeffizienten in die Slant-Koeffizienten. Eine Orthogonal-Transformation, die ähnlich einfach berechnet werden kann, ist die Haar-Transformation. Deshalb liegt es nahe, sie zur Berechnung der Slant-Transformation heranzuziehen. Die Slant-Transformation von Fino und Algazi [4.38] basiert auf der HT und kombiniert Elemente der Haar-Matrix mit der Slant-Transformation. Der Algorithmus benötigt r+2 Iterationen, eine mehr als der Walsh-Slant-Algorithmus, jedoch sind weniger Operationen erforderlich. Allerdings werden die Koeffizienten in ungeordneter Form erzeugt. Ein alternativer Algorithmus der ebenfalls von Fino und Algazi [4.38] angegeben wurde, ergibt eine natürliche Koeffizientenordnung, jedoch sind zusätzliche Speicherplätze für Zwischenergebnisse notwendig.

5 Berechnungsstrategien zweidimensionaler Transformationen

Zweidimensionale (2D-) Transformationen sind wichtige Hilfsmittel für die digitale Signalverarbeitung zweidimensionaler Signale, z.B. für die Filterung, Rekonstruktion und Kodierung von Bildern, für die Mustererkennung und für andere Aspekte der digitalen Bildverarbeitung. Obgleich die hier behandelten Methoden und Algorithmen i. allg. auch höhere Dimensionen zulassen, beschränken wir uns in diesem Buch auf zweidimensionale Transformationen. Höhere Dimensionen treten in der Signalverarbeitung seltener auf. Einige der in Kapitel 6 vorgestellte Algorithmen können zudem ohne große Schwierigkeiten für höhere Dimensionen erweitert werden.

Es erscheint sinnvoll, an dieser Stelle auf die Mehrdeutigkeit des Begriffs *Dimension* im Zusammenhang mit diskreten Orthogonaltransformationen hinzuweisen. Im Abschnitt 3.4.1 wurde die Aufteilung der Algorithmen in Module dargestellt. Dabei hat jedes Modul seinen von anderen unabhängigen Laufindex. So gesehen, besäße der Algorithmus so viele Dimensionen wie er Module hat. Im Fall einer Radix-p-Transformation läge also die Dimension $\log_p N$ vor. Die Effektivität eines Radix-p- oder eines Primfaktor-Algorithmus' beruht demnach auf der Zerlegung eines eindimensionalen Problems in ein mehrdimensionales. In diesem Kapitel betrachten wir jedoch eine andere Auslegung des Begriffs der Dimensionalität, nämlich diejenige des zu transformierenden Signals. Wir vermeiden außerdem, die Ordnung von Matrizen und Vektoren mit dem Ausdruck *Dimension* zu belegen. Wir betrachten also Daten-Arrays, die *a priori* zweidimensional, d.h. als Matrix gegeben sind, und in einen ebenfalls zweidimensionalen Satz von Koeffizienten transformiert werden sollen. Die 2D-Koeffizienten geben an, mit welchem Anteil die zugehörige 2D-Basisfunktion (Basisbild) in der Orthogonalzerlegung der 2D-Funktion enthalten ist (vgl. Bild 1.1). Als Beispiel illustriert Bild 5.1 die 64 Basisbilder der 2D-Walsh-Transformation (in Hadamard-Ordnung) eines (8×8)-Arrays.

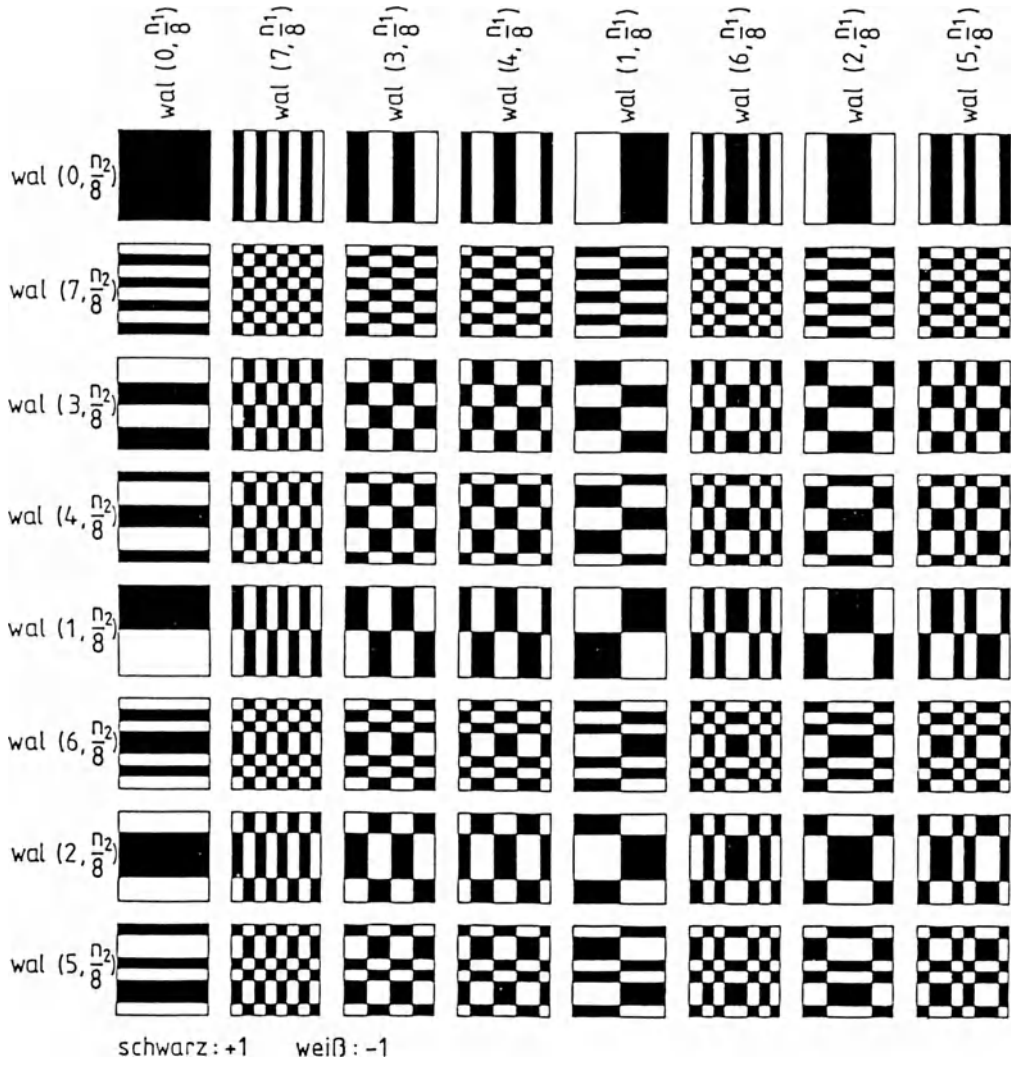


Bild 5.1. Basisbilder der 2D-Walsh-Hadamard-Transformation

Für die Durchführung einer 2D-Transformation kann man i.allg. unter verschiedenen Strategien wählen. Neben der häufig benutzten Separierung in Zeilen- und Spaltentransformationen (Abschnitt 5.2) bestehen i. allg. weitere Möglichkeiten, die ggf. mit Vorteil zu benutzen sind. So ist es z.B. möglich, die 2D-Datenmenge direkt zu transformieren (Abschnitt 5.3) oder die Zeilen- und Spaltentransformationen zu einer einzigen Transformation zusammenzufassen (Abschnitt 5.1). Ähnlich wie bei den eindimensionalen Transformationen kann man u.U. die Koeffizienten aus Koeffizienten niedrigerer Ord-

nung oder aus den Koeffizienten einer anderen Transformation (Abschnitt 5.4) berechnen. Speziell für die digitale Bildverarbeitung ist eine Strategie interessant, bei der die Koeffizienten aus Koeffizienten der nächstniedrigerer Ordnung berechnet werden, wobei ausschließlich Koeffizienten von Unterbilder des aktuellen (Teil-) Bildes benutzt werden. Wir nennen dies eine hierachische Berechnung von Koeffizienten (vgl. Abschnitt 5.4).

Die beiden wichtigsten Strategien bei der Berechnung zweidimensionaler Transformationen werden durch die Bilder 5.2 und 5.3 anhand eines (4×4)-

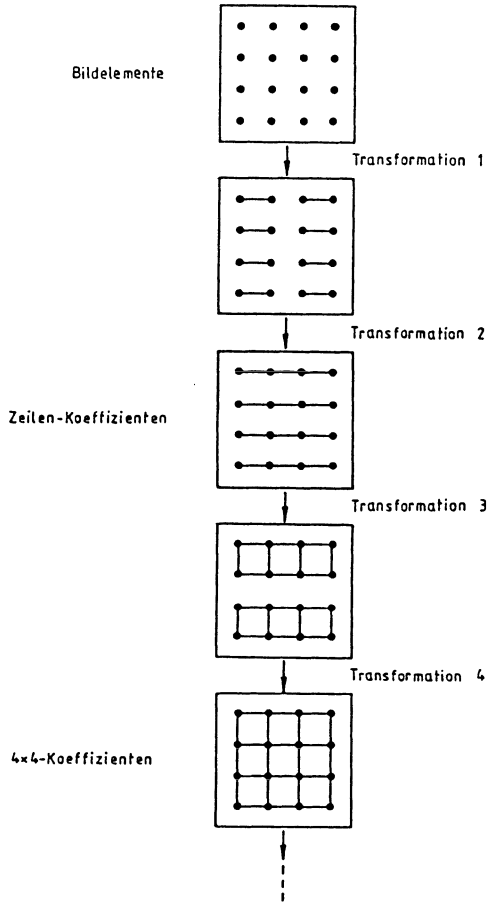


Bild 5.2. Schematische Darstellung einer 2D-Transformation unter Benutzung der Zeilenkoeffizienten

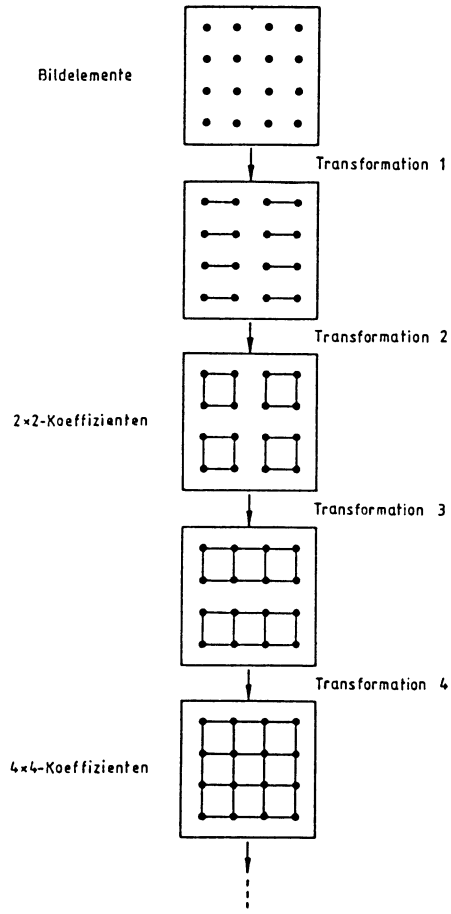


Bild 5.3. Schematische Darstellung einer diskreten (hierarchischen) 2D-Transformation

Bildsignals schematisch dargestellt. Bei der Strategie gemäß Bild 5.2 werden aus den 2D-Datenelementen zunächst durch die Transformationen 1 und 2 die Koeffizienten der Bildzeilen gebildet. Die Transformationen 3 und 4 verarbeiten diese zu den Koeffizienten des (4×4)-Arrays. Im Gegensatz hierzu zeigt Bild 5.3 ein Beispiel für die direkte Berechnung von 2D-Koeffizienten. Dabei werden zunächst die Koeffizienten von (2×2)-Unterbildern berechnet und aus diesen dann die Koeffizienten des (4×4)-Arrays. Wir weisen jedoch darauf hin, daß die im Bild 5.3 dargestellte hierarchische Koeffizientengenerierung auf die $(WHT)_H$ anwendbar ist, auf die DFT dagegen nicht. Die 2D-DFT kann je-

doch in einer ähnlichen Weise direkt, d.h. ohne Benuztung der Zeilen- bzw. Spaltenkoeffizienten gewonnen werden (vgl. Abschnitt 5.3).

Eine auf der Menge der geordneten Paare reeller Werte $(n_1 n_2)$ definierte reelle oder komplexwertige Funktion x heißt periodisch mit den Perioden N_1 und N_2 $(N_1, N_2 > 0)$, falls

$$x(n_1 + N_1, n_2) = x(n_1, n_2), \quad \text{und auch} \quad x(n_1 - N_1, n_2) = x(n_1, n_2),$$

sowie

$$x(n_1, n_2 + N_2) = x(n_1, n_2), \quad \text{und auch} \quad x(n_1, n_2 - N_2) = x(n_1, n_2)$$

für alle endlichen reellen Zahlen n_1, n_2 gilt. Wir betrachten hier ausschließlich ganzzahlige n_1, n_2.

5.1 Berechnung der 2D-Koeffizienten durch eine 1D-Transformation (und umgekehrt)

Von mehreren Autoren [5.1], [5.2] wurde beobachtet, daß eine lexikographische Umordnung der Elemente einer (N×N)-Datenmatrix ($N=2^r$, $r=1,2,\ldots$) in einen Vektor der Ordnung 2^{2r} die Überführung einer 2D-Transformation in eine eindimensionale Transformation erlaubt. Sofern ausreichend Speicherplatz für die 2D-Datenmenge zur Verfügung steht, läßt sich durch eine solche Maßnahme eine Vereinfachung des Transformationsalgorithmus' erzielen. Außerdem können u.U. auch Einsparungen bei der Anzahl der Multiplikationen erzielt werden [5.2]. Umgekehrt besteht immer dann das Bedürfnis, eine 1D-Transformation als eine 2D-Transformation zu berechnen, wenn der Arbeitsspeicher zwar die Berechnung einer Transformation der Länge N, nicht aber der gewünschten Länge N^2 zuläßt. Die Bedingungen unter denen derartige Überführungen zwischen 1D- und 2D-Transformationen möglich sind, wurden von Arazi [5.1] untersucht und sind Gegenstand dieses Abschnitts.

Für die lexikographische Ordnung der Elemente einer (N×N)-Matrix in einen Vektor der Ordnung $N^2 = 2^{2r}$ hat man verschiedene Möglichkeiten. Für N = 4×4 zeigt Bild 5.4 einige Beispiele möglicher Aneinanderreihungen. Ausgehend von der zeilenweisen Aneinanderreihung kann man nach Tabelle 3.1 verschiedene BEO-Umordnungen benutzen. Ob allerdings durch die lexikographischen Einord-

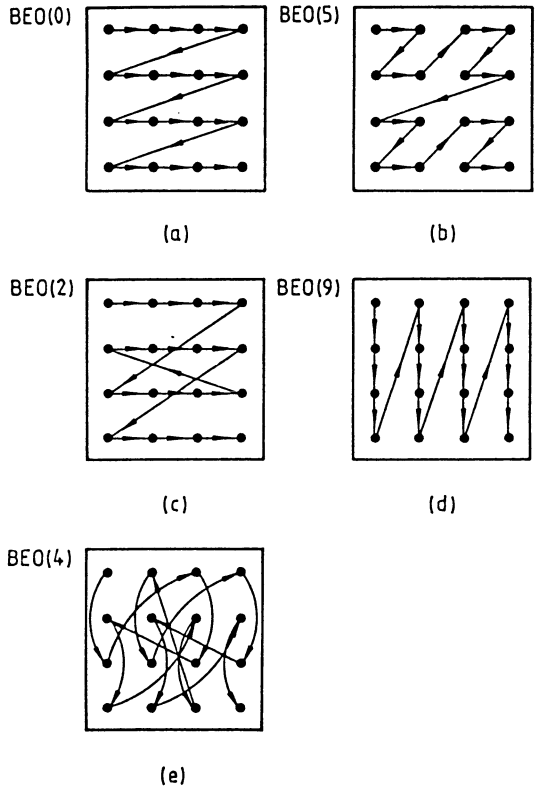


Bild 5.4. Lexikographische Ordnung der Elemente einer (4×4)-Matrix gemäß Tabelle 3.1: a) BEO(0), b) BEO(5), c) BEO(2), d) BEO(9), e) BEO(4)

nungen der Matrixelemente in einen Vektor tatsächlich die Erzeugung der 2D-Koeffizienten einer bestimmten Transformation erreicht wird, muß später näher untersucht werden (vgl. Abschnitt 5.3).

Zunächst wird die zeilenweise Einordnung der Elemente einer Datenmatrix in einen Vektor betrachtet [5.1]. Dazu bildet man einen Vektor $\underline{x}$ der Ordnung N^2 aus einer Datenmatrix [x] der Ordnung N×N, wobei das Matrixelement x_{ij} zum Vektorelement x_{iN+j} wird, i,j = 0,1,...,N-1. Es ist nun zu untersuchen, ob und unter welchen Umständen

$$\underline{X} = [T]\ \underline{x} \tag{5.1}$$

und gleichzeitig

$$[X] = [S][x][S] \tag{5.2}$$

gilt. Dabei ist [T] eine 1D-Transformationsmatrix der Ordnung $2^{2r} \times 2^{2r}$ und [S] eine (symmetrische) Transformationsmatrix der Ordnung $2^r \times 2^r$.

Die Beziehungen (5.1) und (5.2) sind genau dann gleichzeitig erfüllt, wenn

$$[T] = [S] \otimes [S] \tag{5.3}$$

gilt [5.1], wobei das Symbol $\otimes$ das Kronecker-Produkt bezeichnet (vgl. Abschnitt 3.3). In diesem Fall gilt $[S][S] = \pm m \cdot [I]$ genau dann, wenn $[T][T] = m^2 \cdot [I]$ mit $m = 1,2,\ldots$. Die Beziehung (5.3) ist für die Walsh-Hadamard-Transformation erfüllt, nicht jedoch für Fourier-Transformation. Daraus folgt, daß die Koeffizienten einer 2D-$(WHT)_H$ der Ordnung $2^r \times 2^r$ über eine 1D-$(WHT)_H$ der Länge 2^{2r} berechnet werden können und umgekehrt, wobei der Datenvektor durch Konkatenation der Zeilen der Datenmatrix erzeugt wird. Entsprechendes gilt nicht für die DFT, denn (5.3) wird von Fourier-Matrizen nicht erfüllt. Auch wenn die beiden Matrizen [S] in (5.3) durch irgendwelche andere Matrizen der Ordnung $2^r \times 2^r$ ersetzt werden, ist eine zweidimensionale Berechnung einer 1D-DFT nicht möglich [5.1]. Umgekehrt kann man eine Matrix [T] angegeben, mit deren Hilfe man eine 2D-DFT der aneinandergereihten Datenzeilen eindimensional berechnen kann (vgl. (5.3)). Die Matrix [S] ist in diesem Fall eine Fourier-Matrix, während die Transformationsmatrix [T] für die eindimensionale Transformation keine Fourier-Matrix ist. Als Beispiel betrachten wir den Fall N=2. Dafür gilt

$$[S(1)] = \begin{bmatrix} 1 & 1 \\ 1 & V \end{bmatrix} \quad \text{und} \quad [T(1)] = [S(1)] \otimes [S(1)] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & V & 1 & V \\ 1 & 1 & V & V \\ 1 & V & V & V^2 \end{bmatrix}.$$

mit $V = \exp(2\pi j/2) = -1$. Man beachte, daß [T(1)] reell ist, während eine Fourier-Matrix gleicher Ordnung auch imaginäre Elemente enthält.

Die hier beschriebene Prozedur benutzt die zeilenweise Aneinanderreihung der Matrixzeilen, um den Datenvektor zu bilden. Man kann zeigen, daß auch die anderen im Bild 5.4 dargestellten Konkatenationen bei der $(WHT)_H$ zulässig

sind. Dies folgt aus der Tatsache, daß bei der $(WHT)_H$ keine Multiplikationen in den Knoten des Signalflußgraphen vorgenommen werden. In Abschnitt 5.3 wird dargelegt, daß eine Anordnung der Matrixelemente im Vektor gemäß Bild 5.4(e) für die direkte Berechnung der Fourier-Koeffizienten besser eignet ist.

5.2 Separierung in Zeilen- und Spaltentransformation

Allgemein kann die diskrete zweidimensionale Transformation eines $(N_1 \times N_2)$-Arrays in der Form

$$X(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1,n_2)g(n_1,n_2,k_1,k_2) \tag{5.4}$$

dargestellt werden. Dabei ist $g(n_1,n_2,k_1,k_2)$ der Transformationskern, der von den Laufindizes n_1 und n_2 des Objektbereichs sowie k_1 und k_2 des Transformationsbereichs abhängig ist. $x(n_1,n_2)$ und $X(k_1,k_2)$ sind die 2D-Objektdaten bzw. die 2D-Transformationskoeffizienten. Für die Rücktransformation gilt entsprechend

$$x(n_1,n_2) = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X(k_1,k_2)g^{-1}(n_1,n_2,k_1,k_2).$$

Bei den in diesem Buch behandelten Orthogonaltransformationen sind die Transformationskerne separierbar, d.h. $g(n_1,n_2,k_1,k_2) = g_1(n_1,k_1)g_2(n_2,k_2)$. Wenn außerdem Symmetrie in Bezug auf die Koordinatenrichtungen vorliegt, ist

$$g_1(n_1,k_1) = g_2(n_2,k_2) = g(n,k).$$

Unter Voraussetzung der Separierbarkeit des Transformationskerns kann die 2D-Transformation in zwei Schritten berechnet werden (vgl. Bild 5.5)

$$X(k_1,k_2) = (N_1N_2)^{-1} \sum_{n_1=0}^{N_1-1} \{ \sum_{n_2=0}^{N_2-1} x(n_1,n_2)g(n_2,k_2) \} g(n_1,k_1). \tag{5.5}$$

Die Reihenfolge der Summenbildung ist beliebig, d.h. es ist gleich, ob zunächst die Koeffizienten der Zeilen gebildet werden, und daraus die 2D-Koeffizienten, oder zuerst die Spaltenkoeffizienten. Selbstverständlich gelten die gleichen Überlegungen auch für die Rücktransformation.

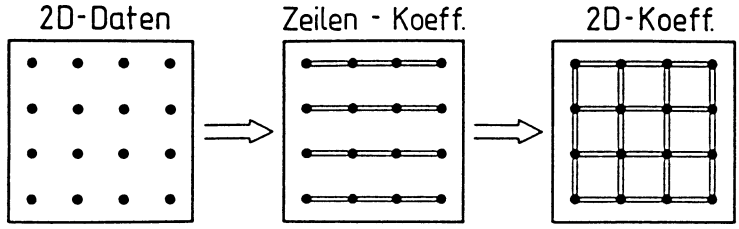


Bild 5.5. Schematische Darstellung des Zeile/Spalte-Algorithmus'

Von (5.5) ausgehend kann man eine Matrixschreibweise einführen. Es gilt nämlich $[x(n_1,n_2)][G(n_2,k_2)] = [Z(n_1,k_2)]$, wobei $[Z]$ eine $(N_1 \times N_2)$-Matrix ist, die die Zeilenkoeffizienten enthält. Die 2D-Koeffizienten erhält man dann aus $[X(k_1,k_2)] = [G(n_1,k_1)][Z(n_1,k_2)]$. Eingesetzt folgt für die 2D-Transformation

$$[X] = [G_1][x][G_2], \tag{5.6}$$

wobei $[X]$ und $[x]$ die Koeffizientenmatrix bzw. die Datenmatrix bezeichnen. Die Matrizen $[G_1]$ und $[G_2]$ sind die Transformationsmatrizen für die Spalten- bzw. Zeilentransformationen. Sie haben die Größen $N_2 \times N_1$ bzw. $N_1 \times N_2$. Entsprechend gilt für die Rücktransformation

$$[x] = [G_1^{-1}][X][G_2^{-1}], \tag{5.7}$$

wobei der Faktor $(N_1N_2)^{-1}$ noch einzubeziehen ist. Die Gleichungen (5.6) und (5.7) setzen natürlich nichtsinguläre Transformationsmatrizen voraus, was im Fall der Orthogonaltransformation *a priori* gegeben ist. Für quadratische Datenarrays ($N_1 = N_2$) wird $[G_1]=[G_2]=[G]$ und $[G_1^{-1}]=[G_2^{-1}]=[G^1]$.

Als Beispiel für eine zweidimensionale Transformation durch den Zeile/Spalte-Algorithmus benutzen wir einen einfachen Radix-2-Algorithmus für 16 Punkte, um die DFT eines (4×4)-Arrays zu berechnen. Im Bild 5.6(a) ist dies anhand eines Signalflußgraphen illustriert. Die beiden ersten Iterationen generieren die Koeffizientensätze der Zeilen, die zu den Koeffizienten des gesamten (4×4)-Arrays weiterverarbeitet werden (vgl. Bild 5.6(b)). Man beachte, daß

(1) die Radix-2-Verknüpfungen gemäß der Reihenfolge der Basisbilder von Bild 5.14 gewählt sind, und

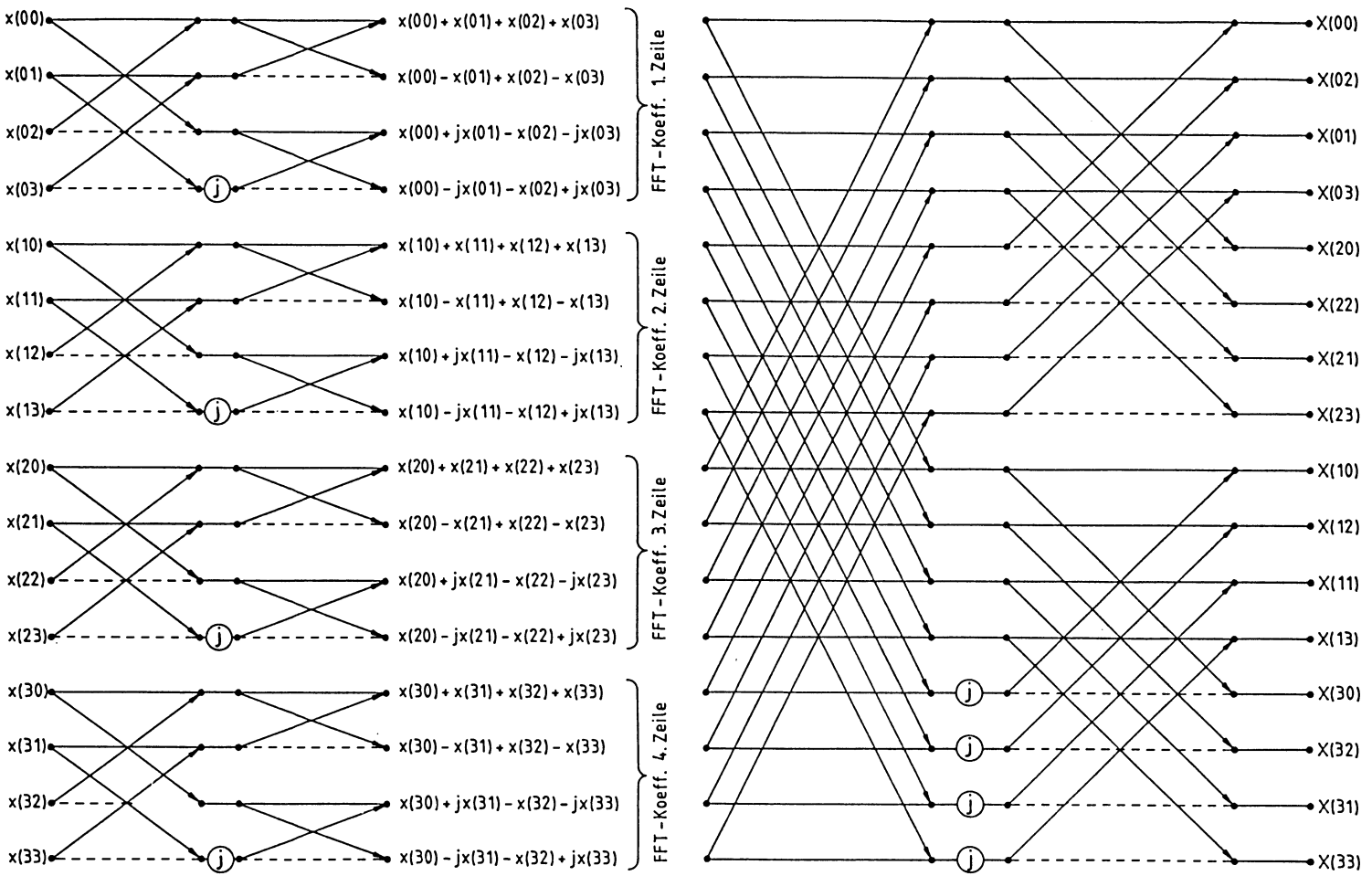


Bild 5.6(a). SFG der FFT eines (4×4)-Arrays für den Zeile/Spalte-Algorithmus

```
X(00) = x(00)+ x(01)+x(02)+ x(03)+ x(10)+ x(11)+ x(12)+ x(13)+x(20)+ x(21)+ x(22)+ x(23)+ x(30)+ x(31)+ x(32)+ x(33)
X(02) = x(00)- x(01)+x(02)- x(03)+ x(10)- x(11)+ x(12)- x(13)+x(20)- x(21)+ x(22)- x(23)+ x(30)- x(31)+ x(32)- x(33)
X(01) = x(00)+jx(01)-x(02)-jx(03)+ x(10)+jx(11)- x(12)-jx(13)+x(20)+jx(21)- x(22)-jx(23)+ x(30)+jx(31)- x(32)-jx(33)
X(03) = x(00)-jx(01)-x(02)+jx(03)+ x(10)-jx(11)- x(12)+jx(13)+x(20)-jx(21)- x(22)+jx(23)+ x(30)-jx(31)- x(32)+jx(33)
X(20) = x(00)+ x(01)+x(02)+ x(03)- x(10)- x(11)- x(12)- x(13)+x(20)+ x(21)+ x(22)+ x(23)- x(30)- x(31)- x(32)- x(33)
X(22) = x(00)- x(01)+x(02)- x(03)- x(10)+ x(11)- x(12)+ x(13)+x(20)- x(21)+ x(22)- x(23)- x(30)+ x(31)- x(32)+ x(33)
X(21) = x(00)+jx(01)-x(02)- x(03)- x(10)-jx(11)+ x(12)+jx(13)+x(20)+jx(21)- x(22)-jx(23)- x(30)-jx(31)+ x(32)+jx(33)
X(23) = x(00)-jx(01)-x(02)+jx(03)- x(10)+jx(11)+ x(12)-jx(13)+x(20)-jx(21)- x(22)+jx(23)- x(30)+jx(31)+ x(32)-jx(33)
X(10) = x(00)+ x(01)+x(02)+ x(03)+jx(10)+jx(11)+jx(12)+jx(13)-x(20)- x(21)- x(22)- x(23)-jx(30)-jx(31)-jx(32)-jx(33)
X(12) = x(00)- x(01)+x(02)- x(03)+jx(10)-jx(11)+jx(12)-jx(13)-x(20)+ x(21)- x(22)+ x(23)-jx(30)+jx(31)-jx(32)+jx(33)
X(11) = x(00)+jx(01)-x(02)-jx(03)+jx(10)- x(11)-jx(12)+ x(13)-x(20)-jx(21)+ x(22)+jx(23)-jx(30)+ x(31)+jx(32)- x(33)
X(13) = x(00)-jx(01)-x(02)+jx(03)+jx(10)+ x(11)-jx(12)- x(13)-x(20)+jx(21)+ x(22)-jx(23)-jx(30)- x(31)+jx(32)+ x(33)
X(30) = x(00)+ x(01)+x(02)+ x(03)-jx(10)-jx(11)-jx(12)-jx(13)-x(20)- x(21)- x(22)- x(23)+jx(30)+jx(31)+jx(32)+jx(33)
X(32) = x(00)- x(01)+x(02)- x(03)-jx(10)+jx(11)-jx(12)+jx(13)-x(20)+ x(21)- x(22)+ x(23)+jx(30)-jx(31)+jx(32)-jx(33)
X(31) = x(00)+jx(01)-x(02)-jx(03)-jx(10)+ x(11)+jx(12)- x(13)-x(20)-jx(21)+ x(22)+jx(23)+jx(30)- x(31)-jx(32)+ x(33)
X(33) = x(00)-jx(01)-x(02)+jx(03)-jx(10)- x(11)+jx(12)+ x(13)-x(20)+jx(21)+ x(22)-jx(23)+jx(30)+ x(31)-jx(32)- x(33)
```

Bild 5.6(b). (4×4)–FFT–Koeffizienten

(2) im Gegensatz zur direkten 2D-Transformation bereits nach der 2.Iteration komplexe Werte auftreten.

Vorteilhaft bei den Zeile/Spalte-Algorithmen ist die Tatsache, daß nur ein eindimensionaler Algorithmus benötigt wird, der zunächst auf die Zeilensequenzen und danach spaltenweise auf die Zeilen-Koeffizienten angewandt wird, bzw. umgekehrt. Das Verfahren erfordert aber i. allg. nach den ersten N 1D-Transformationen eine Transponierung der Matrix, die die 1D-Koeffizienten enthält. Obwohl es Algorithmen für effiziente Transponierungen gibt [5.3], wird durch die Transponierung der o.g. Vorteile u.U. aufgehoben oder gar ins Gegenteil verkehrt. Algorithmen, die ohne Transponierung auskommen (vgl. Abschnitt 5.3), sind deshalb von großem Interesse.

5.3 Direkte Berechnung der 2D-Koeffizienten aus der 2D-Datenmenge

Die Notwendigkeit einer Transponierung der 1D-Koeffizientenmatrix, die den Zeile/Spalte-Algorithmen anhaftet, hat zur Entwicklung verschiedener Algorithmen geführt, die diesen Nachteil vermeiden. Bei diesen werden die 2D-Koeffizienten aus der 2D-Datenmenge direkt, d.h. ohne Benutzung von Zeilen- bzw. Spaltenkoeffizienten berechnet. Das Grundprinzip dieser Methoden besteht darin, das (N×N)-Datenfeld zunächst geeignet umzuordnen, hiervon die Koeffizienten aller (p×p)-Untermatrizen zu berechnen und diese rekursiv zu Koeffizientensätzen von $(p^i \times p^i)$-Untermatrizen zu verbinden, bis $p^i = N$ erreicht ist ($i = 2,3,\ldots$). Indem man den Koeffizientensatz des gesamten 2D-Arrays rekursiv aus den Koeffizienten von Subarrays aufbaut, kann die Transponierung geschickt in die einzelnen Iterationsschritte integriert werden. Darüberhinaus benötigen diese Algorithmen i. allg. weniger Multiplikationen mit Konstanten. Als Beispiele betrachten wir in diesem Abschnitt die 2D-$(WHT)_H$ und die 2D-FFT. Einzelheiten spezieller Algorithmen werden im Kapitel 6 ausführlich dargestellt.

Die Algorithmen zur direkten Berechnung von 2D-Transformationen berücksichtigen die zweidimensionale Anordnung der Daten unmittelbar. Bild 5.7 illustriert dies an einem Beispiel. Ein (8×8)-Datenarray $\{x_{ij}\}$ sei nach der i-ten Zeile und der j-ten Spalte indiziert. Wir stellen uns das Array in vier

$$[x(3)] = \left[\begin{array}{cc|cc|cccc} x(00) & x(01) & x(02) & x(03) & x(04) & x(05) & x(06) & x(07) \\ x(10) & x(11) & x(12) & x(13) & x(14) & x(15) & x(16) & x(17) \\ \hline x(20) & x(21) & x(22) & x(23) & x(24) & x(25) & x(26) & x(27) \\ x(30) & x(31) & x(32) & x(33) & x(34) & x(35) & x(36) & x(37) \\ \hline x(40) & x(41) & x(42) & x(43) & x(44) & x(45) & x(46) & x(47) \\ x(50) & x(51) & x(52) & x(53) & x(54) & x(55) & x(56) & x(57) \\ x(60) & x(61) & x(62) & x(63) & x(64) & x(65) & x(66) & x(67) \\ x(70) & x(71) & x(72) & x(73) & x(74) & x(75) & x(76) & x(77) \end{array}\right]$$

Bild 5.7. Hierarchische Aufteilung eines (8×8)-Arrays

(4×4)-Arrays aufgeteilt vor. Diese seien wiederum jeweils in vier (2×2)-Arrays aufgeteilt, bis man schließlich vier einzelne Datenwerte erhält. Umgekehrt kann man sich das (8×8)-Array aufgebaut denken, indem zunächst sechzehn (2×2)-Arrays gebildet werden, aus diesen vier (4×4)-Arrays und daraus ein (8×8)-Array. Die Datenelemente x_{ij} können ggf. durch eine Umsortierung aus der ursprünglichen 2D-Datenmenge hervorgegangen sein (z.B. im Fall der DFT). Wir verwenden außerdem die 1D-Darstellung der 2D-Transformationen gemäß Abschnitt 5.1.

Wir betrachten zunächst die direkte 2D-Walsh-Hadamard-Transformation

$$[X_H(L)] = [H_H(L)][x(L)][H_H(L)] \tag{5.8}$$

mit $[H_H(L)] = [H_H(1)] \otimes [H_H(L-1)]$, $L = 1,2,\ldots,N$ und $N=2^r$.

Die Datenmatrix $[x(L)]$ wird in vier Untermatrizen $[x_i(L-1)]$ aufgeteilt, $i = 0,1,2,3$

$$[x(L)] = \begin{bmatrix} [x_0(L-1)] & [x_1(L-1)] \\ [x_2(L-1)] & [x_3(L-1)] \end{bmatrix} \tag{5.9}$$

und in einen Vektor $\underline{x}(L)$ eingeordnet

$$\underline{x}(L) = [\underline{x}_0^T(L-1),\ \underline{x}_1^T(L-1),\ \underline{x}_2^T(L-1),\ \underline{x}_3^T(L-1)]^T. \tag{5.10}$$

Man beachte, daß die Vektoren 2^{2r} Elemente besitzen. Durch iterative Anwendung von (5.9) und (5.10) erhält man schließlich den geordneten Datenvektor, z.B. für die Datenmatrix im Bild 5.7

$$\underline{x}(3) = [x_{00}, x_{01}, x_{10}, x_{11}, x_{02}, x_{03}, x_{12}, x_{13}, x_{20}, \ldots, x_{22},$$

$$\ldots, x_{04}, \ldots\ldots\ldots, x_{37}, x_{40}, \ldots, x_{73}, x_{44}, \ldots, x_{77}].$$

Es ist leicht zu sehen, daß die unterteilte Datenmatrix (5.9) ein Äquivalent im Transformationsbereich der $(WHT)_H$ hat

$$[X(L)] = \begin{bmatrix} [X_{11}(L-1)] & [X_{12}(L-1)] \\ [X_{21}(L-1)] & [X_{22}(L-1)] \end{bmatrix}, \qquad (5.11)$$

wobei $[X_{11}(L-1)] = [X_{WH0}(L-1)] + [X_{WH1}(L-1)] + [X_{WH2}(L-1)] + [X_{WH3}(L-1)]$

$[X_{12}(L-1)] = [X_{WH0}(L-1)] - [X_{WH1}(L-1)] + [X_{WH2}(L-1)] - [X_{WH3}(L-1)]$

$[X_{21}(L-1)] = [X_{WH0}(L-1)] + [X_{WH1}(L-1)] - [X_{WH2}(L-1)] + [X_{WH3}(L-1)]$

$[X_{22}(L-1)] = [X_{WH0}(L-1)] + [X_{WH1}(L-1)] - [X_{WH2}(L-1)] - [X_{WH3}(L-1)].$

Unter Benutzung der Vektorschreibweise (5.10) kann diese Matrixgleichung umgeformt werden. Mit (5.8) bis (5.11) erhält man

$$\underline{X} = [H_H(1)] \otimes [I(L-1)]\, \underline{X}_{WH}(L),$$

wobei $\underline{X}_{WH}(L) = \{ [\underline{X}_{WH0}(L-1)]^T, [\underline{X}_{WH1}(L-1)]^T, [\underline{X}_{WH2}(L-1)]^T, [\underline{X}_{WH3}(L-1)]^T \}^T$

die in einen Vektor eingeordneten 2D-Koeffizienten der Ordnung L-1 sind. Damit ist gezeigt, wie sich die $(WHT)_H$-Koeffizienten L-ter Ordnung aus den Koeffizienten (L-1)-ter Ordnung bestimmen lassen.

Die Koeffizienten des (N×N)-Arrays können also schrittweise aus den Koeffizienten von 2D-Subarrays aufgebaut werden. Aus Gründen der Übersichtlichkeit beschränken wir uns in einem Beispiel auf ein (4×4)-Array, das wie im Bild 5.8 angegeben indiziert sei. Den zugehörigen Signalflußgraph zeigt Bild 5.9. Die oberen vier Datenwerte entsprechen dem (2×2)-Subarray oben links im Bild 5.8. Nach zwei Iterationen ergeben sich in den oberen Speicherstellen die

x(0 0)	x(0 1)	x(0 2)	x(0 3)
x(1 0)	x(1 1)	x(1 2)	x(1 3)
x(2 0)	x(2 1)	x(2 2)	x(2 3)
x(3 0)	x(3 1)	x(3 2)	x(3 3)

Bild 5.8. Indizierung eines (4×4)-Datenarrays

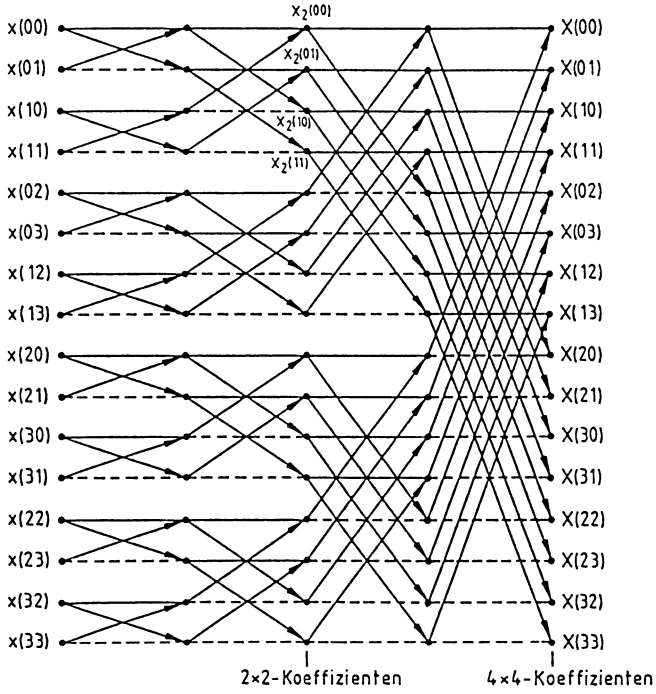


Bild 5.9. SFG der WHT eines (4×4)-Datenarrays für den direkten (hierarchischen) Algorithmus

Koeffizienten $X_2(00)$ bis $X_2(11)$ dieses Subarrays. Entsprechendes gilt für die drei anderen Subarrays. Zur Verdeutlichung des Verfahrens betrachten wir die zugehörigen Basisbilder. Bild 5.10 zeigt die 2D-Basisfunktionen für die vier Koeffizienten $X_2(00)$ bis $X_2(11)$ des ersten Subarrays aus Bild 5.8. Die sechzehn Basisbilder für die (4×4)-$(WHT)_H$-Koeffizienten sind im Bild 5.11

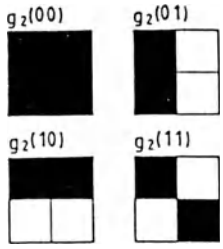


Bild 5.10. (2×2)-Basisbilder $g(k_1,k_2)$ der WHT nach Bild 5.9, schwarz: +1, weiß: −1

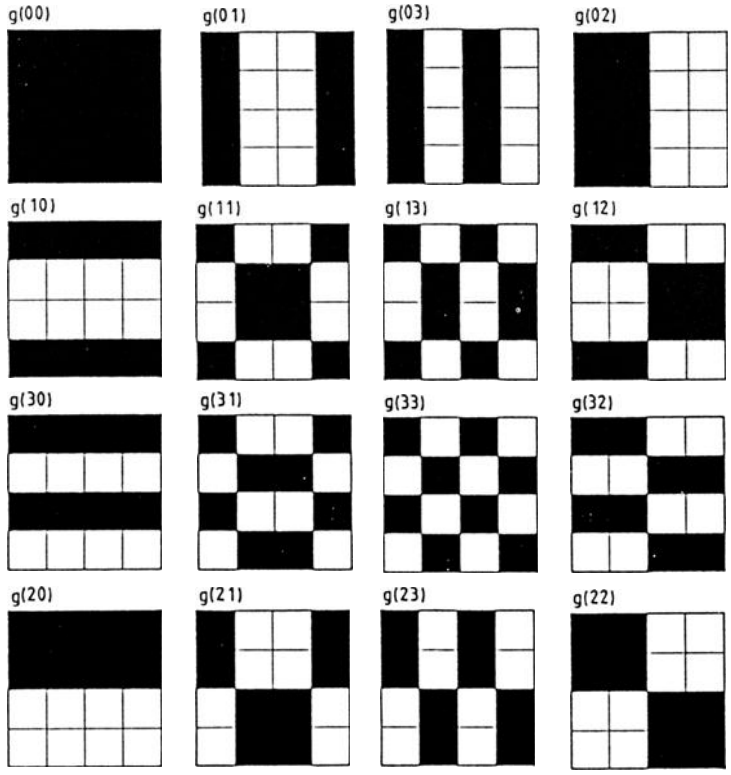


Bild 5.11. (4×4)-Basisbilder $g(k_1,k_2)$ der WHT nach Bild 5.9, schwarz: +1, weiß: −1

dargestellt. Hier ist zu erkennen, wie diese aus den Basisbildern der Subarrays durch Addition bzw. Subtraktion zusammengefügt sind.

Um eine entsprechende Entwicklung für die DFT anzugeben, ist zunächst eine Umordnung der Datenelemente im Objektbereich vorzunehmen. Hierzu schreiben wir die 2D-DFT in Matrixform unter Benutzung der Bitumkehroperation BRO um

[5.4]. Aus $[X_f] = [F][x][F]$, wobei [F] die (symmetrische) Fourier-Matrix ist, folgt mit $[F] = [I_{BRO}][F_{BRO}]$ eine modifizierte Darstellung der 2D-DFT

$$[X_f] = [F_{BRO}]^T[\overline{x}][F_{BRO}] \quad \text{mit} \quad [\overline{x}] = [I_{BRO}][x][I_{BRO}]. \tag{5.12}$$

Im Gegensatz zur ursprünglichen Datenmatrix [x] hat die umgeordnete Matrix $[\overline{x}]$ die Eigenschaft, daß sich aus ihrer Quadrantenzerlegung die 2D-Fourier-Koeffizienten rekursiv errechnen lassen [5.4]. Hierzu führt man eine ähnliche Zerlegungsprozedur von Daten- und Transformationsmatrizen durch, wie oben für die $(WHT)_H$ gezeigt wurde. Dabei ergeben sich wegen der Drehfaktoren etwas kompliziertere Ausdrücke als bei der 2D-$(WHT)_H$.

Wir betrachten nun, wie die Bitumkehroperationen in (5.12) das Datenarray umordnen. Wie man sich leicht anhand der BRO-Definition (vgl. Abschnitte 2.4 und 3.4.3) überzeugt, werden jeweils solche Datenelemente zu (2×2)-Nachbarschaften zusammengefaßt, die ursprünglich um 2^i Schritte in horizontaler und vertikaler Richtung auseinander lagen (i = 1,2...). Bild 5.12 demonstriert wie die Elemente im einfachsten Fall (i=1) zu vertauschen sind. Dabei bedeuten die Doppellinien, daß die verbundenen Elemente durch zwei Schmetterlingsiterationen zusammenzufassen sind. Ein (4×4)-Signalflußgraph, der eine direkte Transformation leistet, ist im Bild 5.13 angegeben. Die Indizierung entspricht der Anordnung gemäß Bild 5.8. Im SFG sind auch die Multiplikationen mit den Drehfaktoren angegeben, deren Ermittlung an dieser Stelle nicht weiter diskutiert wird (vgl. hierzu Abschnitt 6). Wir bemerken jedoch, daß die Multiplikationen nach jeweils zwei Iteration in den Knoten erfolgen, und ihre Anzahl um ein Viertel geringer ist als beim Zeile/Spalte-Algorithmus (vgl. Bild 5.6).

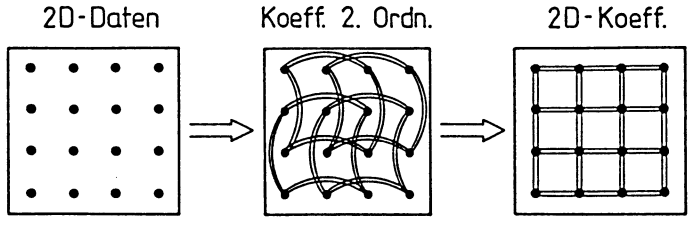


Bild 5.12. Schematische Darstellung einer direkten 2D-Transformation mit umsortierten Objektdaten

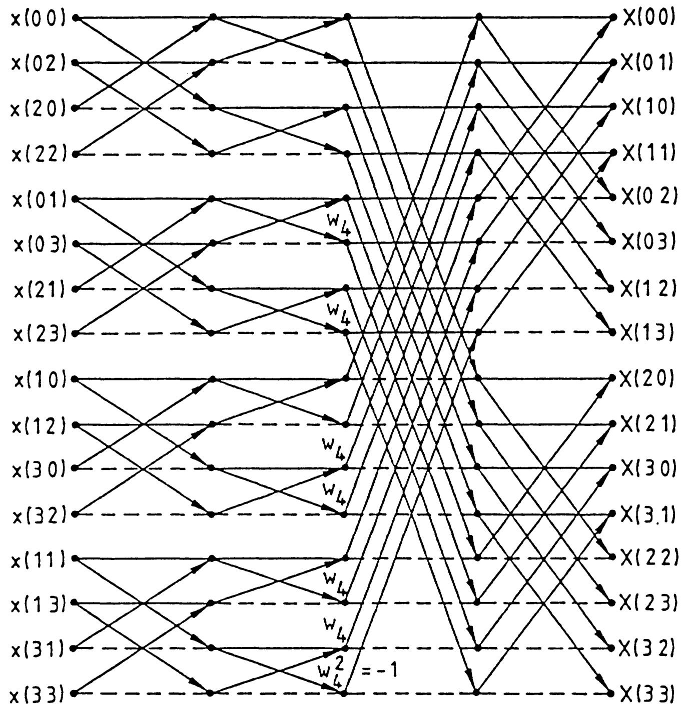


Bild 5.13. SFG der FFT eines (4×4)-Arrays für die direkte 2D-Transformation mit umsortierten Objektdaten

Zum Vergleich mit der 2D-$(WHT)_H$ geben wir auch für dieses Beispiel die Basisbilder an (Bild 5.14). Man beachte, daß die Basisbilder für die (umsortierten) (2×2)-Subarrays gleich denen der (reellen) $(WHT)_H$ sind, d.h. durch Bild 5.10 gegeben sind. Durch Multiplikationen in den Knoten und die folgenden beiden Iterationen werden daraus die (4×4)-Basisbilder gemäß Bild 5.14 (in Übereinstimmung mit dem SFG Bild 5.6) erzeugt. Auf dem in diesem Abschnitt dargestellten Prinzip beruhen eine Reihe von 2D-Algorithmen, die im Kapitel 6 noch ausführlich dargestellt werden ([5.5] bis [5.8]).

5.4 Berechnung aus 2D-Koeffizienten niedrigerer Ordnung

Im Abschnitt 5.3 haben wir ein Verfahren vorgestellt, bei dem die 2D-Koeffizienten aus den Koeffizienten von irgendwelchen durch Umordnung entstandenen 2D-Subarrays gewonnen werden. Für den Fall der 2D-$(WHT)_H$ ergab sich dabei,

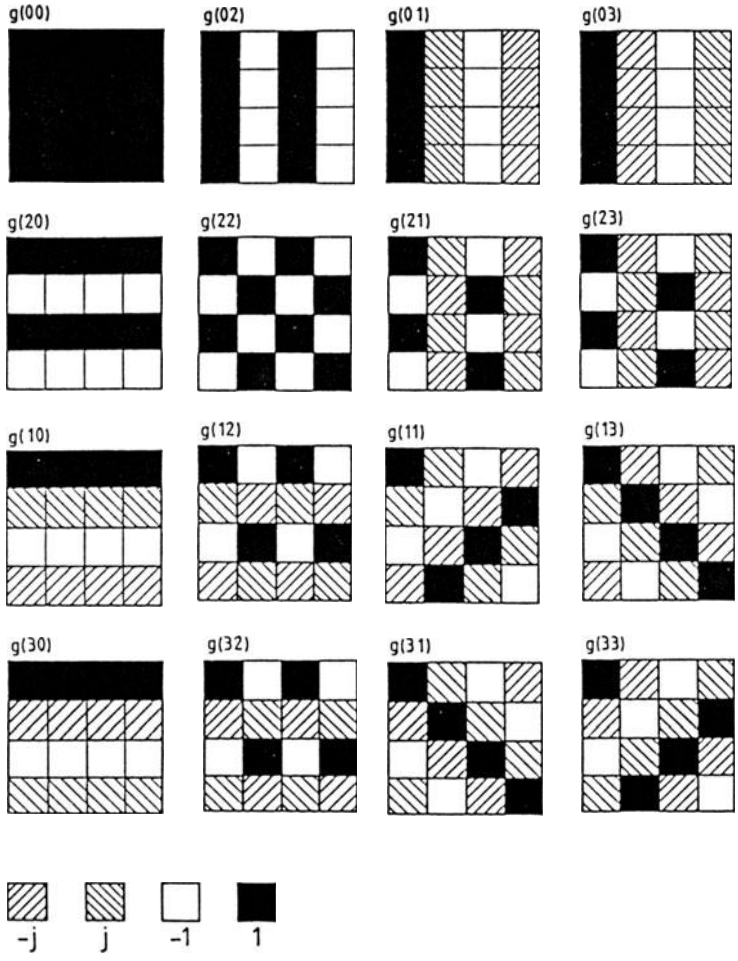


Bild 5.14. (4×4)-Basisbilder $g(k_1,k_2)$ der DFT nach Bild 5.13 und Bild 5.6

daß keine Umordnung benötigt wird und sich die Koeffizienten eines Arrays beliebiger Ordnung aus den Koeffizienten seiner eignen Subarrays bestimmen lassen (vgl. Bild 5.15). Eine solche Transformation nennen wir hierarchisch. Es ist leicht einzusehen, daß dieses Prinzip nur dann anwendbar ist, wenn die Transformationsmatrix durch eine Kronecker-Potenz (vgl. Abschnitt 3.3) darstellbar ist. Deshalb ist die hierarchische Transformation unter den hier behandelten Orthogonaltransformationen auf die $(WHT)_H$ beschränkt.

In der digitalen Bildverarbeitung, speziell bei der Bildkodierung kann es von Vorteil sein, die Transformationsblöcke schrittweise zu vergrößern wie

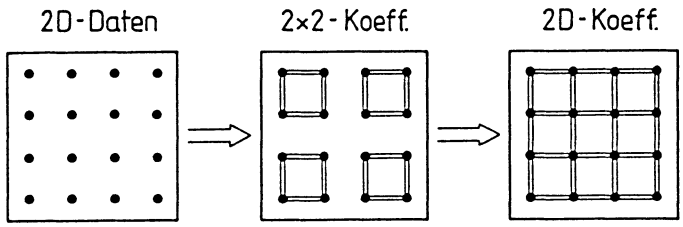


Bild 5.15. Schematische Darstellung einer hierarchischen Transformation

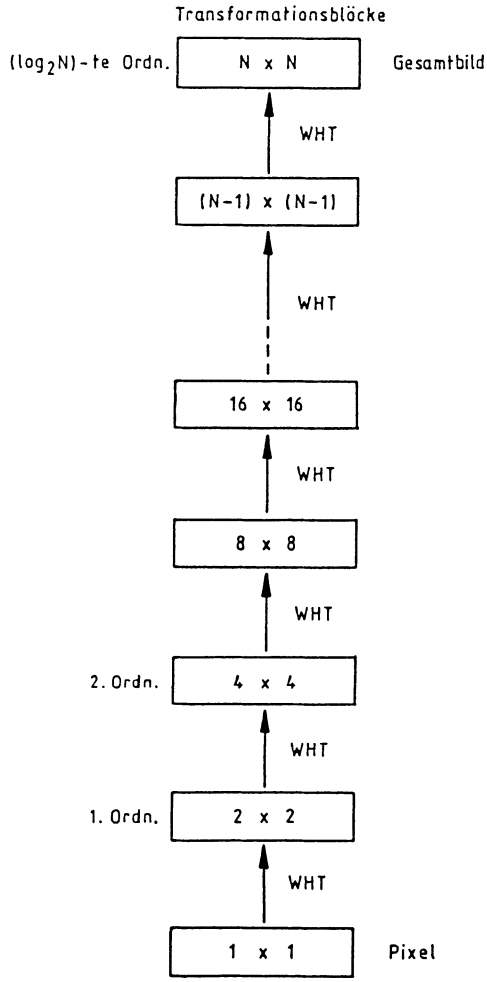


Bild 5.16. Schematische Darstellung von WHT-Transformationsblöcken

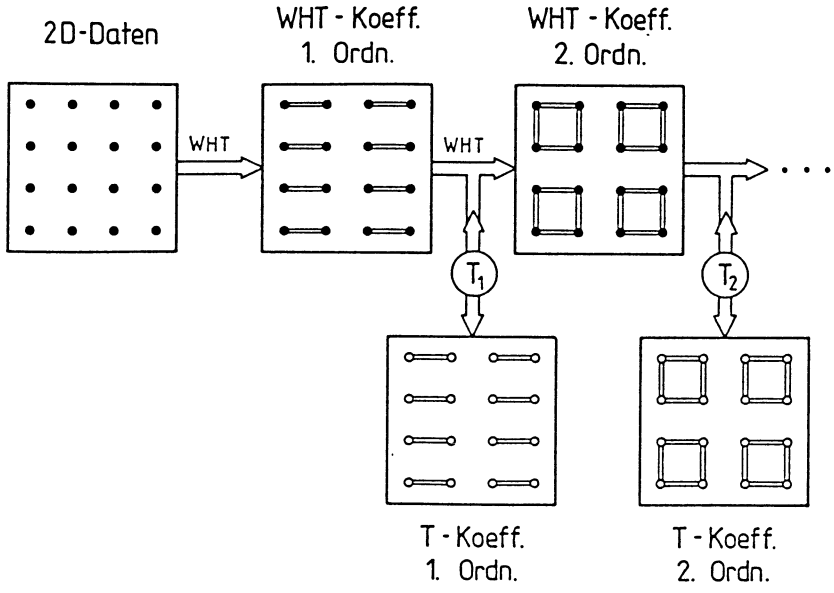


Bild 5.17. Schrittweise Berechnung von T-Koeffizienten über eine hierarchische WHT

es Bild 5.16 zeigt [5.9]. Verwendet man die $(WHT)_H$, so lassen sich die Bildelemente (Pixel) in den Iterationen schrittweise zu Koeffizienten immer größerer Unterbilder verknüpfen. Da der $(WHT)_H$ wegen ihrer einfachen Basisfunktionen gewisse Nachteile anhaften, andere günstigere Transformationen jedoch keinen hierarchischen Aufbau zulassen, bietet sich ein Kompromiß an. Hierfür werden die Daten hierarchisch zu 2D-$(WHT)_H$-Koeffizienten steigender Ordnung verarbeitet, und aus diesen jeweils die 2D-Koeffizienten der gewünschten Transformation errechnet. Bild 5.17 stellt einen solchen Verarbeitungsvorgang schematisch dar. Wir werden im Abschnitt 6.4.2 eine solche Methode vorstellen.

6 Algorithmen zweidimensionaler Transformationen

6.1 Algorithmen für die 2D-Walsh-Transformation

Im Abschnitt 4.1 haben wir die eindimensionale diskrete Walsh-Transformation (DWHT) behandelt und dabei gesehen, daß die Operationen für die DWHT nur aus Additionen bzw. Subtraktionen bestehen. Da keine Multiplikationen benötigt werden, bietet sich die WHT für große Datenmengen, wie z.B. Bilder, besonders an. Außerdem kann man die DWHT auch als Zwischenschritt zur Gewinnung von anderen Transformationen wie z.B. der DFT und der DCT benutzen. Deswegen ist es notwendig, Algorithmen für die zweidimensionale DWHT zu diskutieren.

Die zweidimensionale WHT einer (N×N)-Datenmatrix ist gegeben durch [2.7] [1)]

$$X_{WH}(k_1,k_2) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1,n_2)(-1)^{<n1,k1>+<n2,k2>} \qquad (6.1)$$

mit der inversen Transformation

$$x(n_1,n_2) = N^{-2} \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} X_{WH}(k_1,k_2)(-1)^{<n1,k1>+<n2,k2>} \qquad (6.2)$$

und

$$<n_i,k_i> = \sum_{m=0}^{r-1} V_m U_m, \qquad i = 1,2.$$

Für die nichtnegativen ganzen Zahlen k und n gelten dabei die eindeutigen Dualzahldarstellungen (2.5).

---

1) Aus drucktechnischen Gründen werden Indizes der Exponenten auf der (hochgestellten) Zeile geschrieben.

Es sind verschiedene Algorithmen für die zweidimensionale DWHT entwickelt worden, z.B. der Zeile/Spalte-Algorithmus, der im Abschnitt 5.2 angegeben ist, die Zurückführung auf die eindimensionale DWHT (vgl. Abschnitt 5.1), die hierarchische Transformation usw. In diesem Abschnitt wird die Zurückführung auf eine eindimensionale DWHT nicht weiter diskutiert. Ein Beispiel für die Anwendung dieses Verfahrens auf die WHT (z.B. für den Fall NxN=4x4) ist im Bild 6.1 dargestellt.

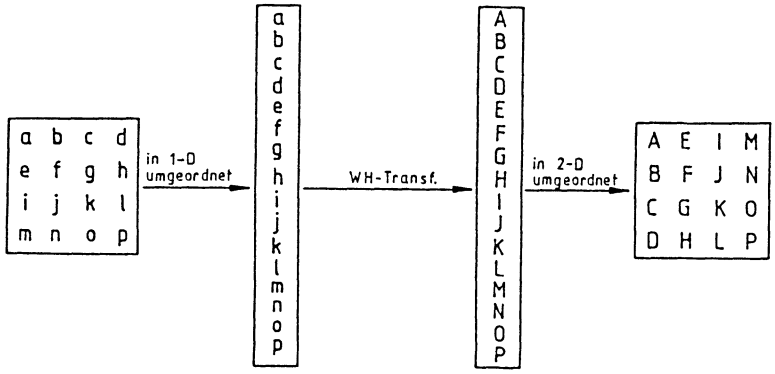


Bild 6.1. 1D-WHT von lexikographisch sortierten 2D-Daten

Es kann zweckmäßig sein, eine 2D-Transformation über die Transformierten von Subarrays zu erzeugen. Für eine Transformation von NxN Elementen könnte man aber auch einen Algorithmus benutzen, wie ihn Bild 5.16 zeigt. Im Fall der WHT ist eine hierarchische Koeffizientenerzeugung möglich (vgl. Abschnitt 5.4), wobei die Koeffizienten jedes Arrays aus den Koeffizienten seiner eigenen Subarrays errechnet werden.

Die konkrete Prozedur der hierarchischen Transformation für die Walsh-Hadamard-Transformation stellt sich wie folgt dar:
Die zu transformierenden Daten $[x(n_1,n_2)]$ seien in Matrixform gegeben:

$$[x(n_1,n_2)] = \begin{bmatrix} x(0,0) & x(0,1) & \dots & x(0,N_2-1) \\ x(1,0) & x(1,1) & \dots & x(1,N_2-1) \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ x(N_1-1,0) & x(N_1-1,1) & \dots & x(N_1-1,N_2-1) \end{bmatrix}.$$

Man spalte sowohl die Datenmatrix als auch die Transformationsmatrix in jeweils vier Quadranten auf:

$$[x(n_1,n_2)] = \begin{bmatrix} [a(n_1,n_2)] & [b(n_1,n_2)] \\ [c(n_1,n_2)] & [d(n_1,n_2)] \end{bmatrix}.$$

Diese vier Teile werden rekursiv weiter in jeweils vier Quadranten aufgeteilt, bis die Ordnung der zu transformierenden Matrizen nur noch 2×2 ist, d.h.

$$x(n_1,n_2) = \begin{cases} a(n_1,n_2) & 0 \leqq n_1,n_2 \leqq (N/2)-1 \\ b(n_1,n_2-(N/2)) & 0 \leqq n_1 \leqq (N/2)-1, \text{ und } (N/2) \leqq n_2 \leqq N-1 \\ c(n_1-(N/2),n_2) & 0 \leqq n_2 \leqq (N/2)-1, \text{ und } (N/2) \leqq n_1 \leqq N-1 \\ d(n_1-(N/2),n_2-(N/2)) & (N/2) \leqq n_1,n_2 \leqq N-1. \end{cases} \tag{6.3}$$

Dabei hat man aus (6.1) und (6.2)

$$\begin{aligned} X_{WH}(k_1,k_2) = & \sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} x(n_1,n_2)\,(-1)^{<n1,k1>+<n2,k2>} \\ & + \sum_{n_1=0}^{(N/2)-1} \sum_{n_2=(N/2)}^{N-1} x(n_1,n_2)\,(-1)^{<n1,k1>+<n2,k2>} \\ & + \sum_{n_1=(N/2)}^{N-1} \sum_{n_2=0}^{(N/2)-1} x(n_1,n_2)\,(-1)^{<n1,k1>+<n2,k2>} \\ & + \sum_{n_1=(N/2)}^{N-1} \sum_{n_2=(N/2)}^{N-1} x(n_1,n_2)\,(-1)^{<n1,k1>+<n2,k2>} \\ = & \sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} abcd(n_1,n_2) \end{aligned} \tag{6.4}$$

mit

$$abcd(n_1,n_2) =$$

$$\{\, a(n_1,n_2) + b(n_1,n_2)(-1)^{<k2,(N/2)>} + c(n_1,n_2)(-1)^{<k1,(N/2)>} +$$

$$+ d(n_1,n_2)(-1)^{<k1,(N/2)>+<k2,(N/2)>} \} (-1)^{<k1,n1>+<k2,n2>}.$$

Im Transformationsbereich ergeben sich folgende Beziehungen:

$$A(k_1,k_2) = X_{WH}(2k_1,2k_2) =$$

$$\sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} \{a(n_1,n_2)+b(n_1,n_2)+c(n_1,n_2)+d(n_1,n_2)\}(-1)^{<k1,n1>+<k2,n2>},$$

$$B(k_1,k_2) = X_{WH}(2k_1,2k_2+1) =$$

$$\sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} \{a(n_1,n_2)-b(n_1,n_2)+c(n_1,n_2)-d(n_1,n_2)\}(-1)^{<k1,n1>+<k2,n2>},$$

$$C(k_1,k_2) = X_{WH}(2k_1+1,2k_2) =$$

$$\sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} \{a(n_1,n_2)+b(n_1,n_2)-c(n_1,n_2)-d(n_1,n_2)\}(-1)^{<k1,n1>+<k2,n2>},$$

$$D(k_1,k_2) = X_{WH}(2k_1+1,2k_2+1) =$$

$$\sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} \{a(n_1,n_2)-b(n_1,n_2)-c(n_1,n_2)+d(n_1,n_2)\}(-1)^{<k1,n1>+<k2,n2>},$$

$$0 \leqq k_1,k_2 \leqq (N/2)-1$$

Damit kann der Zusammenhang zwischen Objekt- und Transformationsbereich gemäß Bild 6.2 verallgemeinert werden.

Wir können auch die Matrixdarstellung zur Erläuterung des Algorithmus' benutzen. Wie bereits mehrfach erwähnt, kann die Hadamard-Matrix höherer Ord-

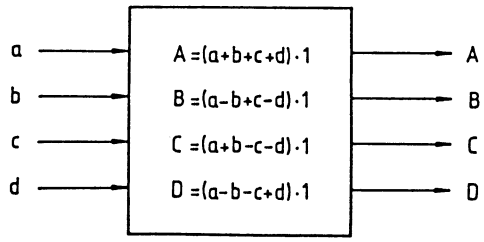


Bild 6.2. Beziehung zwischen Objekt- und Transformationsbereich bei der WHT (vereinfachter Schmetterling)

nung aus Matrizen niedrigerer Ordnung erzeugt werden. Dazu schreiben wir die zweidimensionale DWHT in Matrixform als

$$[X_{WH}(L)] = \begin{bmatrix} [H_H(L-1)] & [H_H(L-1)] \\ [H_H(L-1)] & -[H_H(L-1)] \end{bmatrix} \begin{bmatrix} [Q_{11}(L-1)] & [Q_{12}(L-1)] \\ [Q_{21}(L-1)] & [Q_{22}(L-1)] \end{bmatrix} \begin{bmatrix} [H_H(L-1)] & [H_H(L-1)] \\ [H_H(L-1)] & -[H_H(L-1)] \end{bmatrix}^T .$$

Mit den Abkürzungen $[U_{ij}]$ erhält man

$$[X_{WH}(L)] = \begin{bmatrix} [U_{11}(L-1)] & [U_{12}(L-1)] \\ [U_{21}(L-1)] & [U_{22}(L-1)] \end{bmatrix}. \qquad (6.5)$$

wobei $[U_{11}(L-1)] = [X_{WH11}(L-1)]+[X_{WH12}(L-1)]+[X_{WH21}(L-1)]+[X_{WH22}(L-1)]$,
$[U_{12}(L-1)] = [X_{WH11}(L-1)]-[X_{WH12}(L-1)]+[X_{WH21}(L-1)]-[X_{WH22}(L-1)]$,
$[U_{21}(L-1)] = [X_{WH11}(L-1)]+[X_{WH12}(L-1)]-[X_{WH21}(L-1)]-[X_{WH22}(L-1)]$,
$[U_{22}(L-1)] = [X_{WH11}(L-1)]-[X_{WH12}(L-1)]-[X_{WH21}(L-1)]+[X_{WH22}(L-1)]$.

In dieser Weise kann die zweidimensionale DWHT unter Verwendung der oben beschriebenen Methode hierarchisch ausgeführt werden.

Wie im Kapitel 5 dargelegt wurde, kann eine zweidimensionale DWHT durch drei Arten von Algorithmen ausgeführt werden, nämlich mit dem Zeile/Spalte-Algorithmus, mit dem hierarchischen Algorithmus und durch Zurückführung auf eine eindimensionale DWHT. Wegen des Fehlens von Multiplikatoren (z.B. Drehfaktoren) bei der WHT unterscheiden sich die Signalflußgraphen der Methoden nicht auffallend. Es handelt sich jedoch um unterschiedliche Ansätze, die auch für andere 2D-Transformationen von Bedeutung sind. Zur Veranschaulichung stellen wir hier diese drei Algorithmen für den Fall N×N = 4×4 zusammen:

(1) Der Zeile/Spalte-Algorithmus: Zweimalige Anwendung der eindimensionalen DWHT (Bild 5.5). Dargestellt in der Form der Transformationsmatrizen wird die zweidimensionale DWHT $X_{WH}(k_1,k_2)$ wie folgt ausgeführt:

$$[X_{WH}] = [H_3^2][H_2^2]([x][H_1^2][H_0^2]),$$

Zeilenkoeffizienten: $([x][H_1^2][H_0^2])$,

(4×4)-Koeffizienten: $[X_{WH}]$.

Dabei ist $[H_1^2]$ eine (schwach besetzte) Matrix, die benachbarte Zeilenelemente transformiert, $[H_0^2]$ bildet hieraus die Koeffizienten der 4 Elemente langen Zeilen, $[H_2^2]$ generiert die Koeffizienten von Zeilenpaaren und $[H_3^2]$ erzeugt hieraus die (4×4)-WHT-Koeffizienten. Falls man $[H_3^2]$ $[H_2^2]$ $[x]$ zuerst bildet, erfolgt die 2D-Transformation über die Spaltenkoeffizienten.

(2) Der hierarchische Algorithmus: Gewinnung der 2D-Transformation aus den Transformationskoeffizienten der Unterbilder (vgl. Bild 5.15). Die zweidimensionale DWHT $X_{WH}(k_1,k_2)$ in der Form der Transformationsmatrizen dargestellt ergibt sich dann wie folgt:

$$[X_{WH}] = [H_3^2]([H_2^2][x][H_1^2])[H_0^2],$$

(2×2)-Koeffizienten: $([H_2^2][x][H_1^2])$.

(4×4)-Koeffizienten: $[X_{WH}]$.

(3) Zurückführung auf eine eindimensionale DWHT, d.h. Erzeugung der zweidimensionalen DWHT-Koeffizienten aus den in einen Vektor eingeordneten Daten (vgl. Bild 6.1) Abgesehen von der im Bild 6.1 dargestellten Anordnung gibt es noch weitere Möglichkeiten der Einordnung (vgl. Abschnitt 5.1). Die Rückführung auf eine 1D-Transformation ist wegen der fehlenden Gewichtsfaktoren bei der WHT besonders einfach und kann deshalb auch mit den unter (1) und (2) beschriebenen Methoden verbunden werden. Die 1D-WHT eines zweidimensionalen Datensatzes $[x]$ ist dann (vgl. Abschnitt 5.1)

$$\underline{X}_{WH} = [H_H(4)]\,\underline{x},$$

wobei $[H_H(4)] = [H^4]$ die Hadamard-Matrix der Ordnung 2^4 ist und $\underline{x}$ und $\underline{X}_{WH}$ die lexikographisch geordneten Vektoren des zweidimensionalen Objekt- und des 2D-Transformationsbereichs.

6.2 Algorithmen für die 2D-Fourier-Transformation

Die DFT der zweidimensionalen Funktion $x(n_1,n_2)$ wird als $X_f(k_1,k_2)$ definiert:

$$X_f(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1,n_2) W_{N1}^{k1n1} W_{N2}^{k2n2}, \qquad (6.6)$$

$$k_1 = 0,1,\ldots,N_1-1, \qquad k_2 = 0,1,\ldots,N_2-1.$$

Hierbei ist $x(n_1,n_2)$ Element einer $(N_1 \times N_2)$-Matrix [5.7]

$$[x(n_1,n_2)] = \begin{bmatrix} x(0,0) & x(0,1) & \ldots & x(0,N_2-1) \\ x(1,0) & x(1,1) & \ldots & x(1,N_2-1) \\ \cdot & & & \\ \cdot & & & \\ x(N_1-1,0) & x(N_1-1,1) & \ldots & x(N_1-1,N_2-1) \end{bmatrix}.$$

Die zweidimensionale diskrete Fourier-Transformation ist ein Spezialfall der allgemeinen 2D-Transformation (5.4) mit dem Transformationskern

$$g(n_1,n_2,k_1,k_2) = \exp[-j2\pi((n_1k_1/N_1) + (n_2k_2/N_2))].$$

Gleichung (5.6) entsprechend läßt sich die diskrete 2D-Fourier-Transformierte auch in der Form

$$[X_f] = [\exp(-j2\pi n_1k_1/N_1)][x][\exp(-j2\pi n_2k_2/N_2)],$$

für $n_1,k_1 = 0,1,\ldots,N_1-1, \qquad n_2,k_2 = 0,1,\ldots,N_2-1$

schreiben. Entsprechendes gilt auch für die Rücktransformation. Weil

$$\exp(j2\pi((n_1k_1/N_1) + (n_2k_2/N_2))) = \exp(j2\pi n_1k_1/N_1)\exp(j2\pi n_2k_2/N_2) \qquad (6.7)$$

ist, ergibt sich die inverse diskrete 2D-Fourier-Transformation mit (6.7) in folgender Form

$$[x] = (N_1N_2)^{-1}[\exp(j2\pi k_1n_1/N_1][X_f][\exp(j2\pi k_2n_2/N_2)].$$

In Summenform lautet das Transformationspaar

$$X_f(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1,n_2)\, W_{N1}^{k1n1}\, W_{N2}^{k2n2}$$

$$\updownarrow$$

$$x(n_1,n_2) = (N_1N_2)^{-1} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} X_f(k_1,k_2)\, W_{N1}^{-n1k1}\, W_{N2}^{-n2k2}.$$

Für den Spezialfall $N_1 = N_2 = N$ betrachten wir einige wichtige Eigenschaften der 2D-DFT:

(1) 2D-Periodizität:

$$X_f(k_1,k_2) = X_f(k_1+pN,\ k_2) = X_f(k_1,\ k_2+qN) = X_f(k_1+pN,\ k_2+qN)$$

mit $p, q = \pm1, \pm2, \pm3, \ldots$.

(2) Separierbarkeit: Aus der Definition der 2D-DFT folgt

$$X_f(k_1,k_2) = \sum_{n_1=0}^{N-1} \exp((-j2\pi k_1 n_1)/N) \sum_{n_2=0}^{N-1} x(n_1,n_2)\exp((-j2\pi k_2 n_2)/N). \qquad (6.8)$$

Setzt man

$$x(n_1,k_2) = \sum_{n_2=0}^{N-1} x(n_1,n_2)\exp((-j2\pi k_2 n_2)/N)$$

so kann (6.8) als

$$X_f(k_1,k_2) = \sum_{n_1=0}^{N-1} x(n_1,k_2)\exp((-j2\pi n_1 k_1)/N).$$

geschrieben werden.

(3) 2D-Faltungstheorem:

Die Faltung der beiden zweidimensionalen Funktionen $x_1(n_1,n_2)$ und $x_2(n_1,n_2)$ ist wie folgt definiert

$$x_1(n_1,n_2)**x_2(n_1,n_2) = \sum_{m_1=0}^{N_1-1} \sum_{m_2=0}^{N_2-1} x_1(m_1,m_2)x_2(n_1-m_1,n_2-m_2)$$

$$n_1,n_2 = 0,1,\ldots,N-1,$$

wobei "**" die zweidimensionale Faltungsoperation symbolisiert. Das Faltungstheorem für die 2D-DFT lautet

$$x_1(n_1,n_2)**x_2(n_1,n_2) \longleftrightarrow X_{f1}(k_1,k_2)\cdot X_{f2}(k_1,k_2).$$

6.2.1 Zeile/Spalte-Algorithmus der DFT

Die Idee der Zeile/Spalte-Algorithmen wird anschaulich, wenn man zweidimensionale separierbare Transformationen betrachtet, deren Basisfunktionen Produkte eindimensionaler Basisfunktionen sind (vgl. z.B. Bild 5.11 und Bild 5.14). Wegen der Separierbarkeit der zweidimensionalen DFT läßt sich diese also in zwei eindimensionale DFT zerlegen und durch zweimalige Anwendung der eindimensionalen FFT-Algorithmen darstellen. Dieses Verfahren wird für N=4 durch Bild 6.3 veranschaulicht.

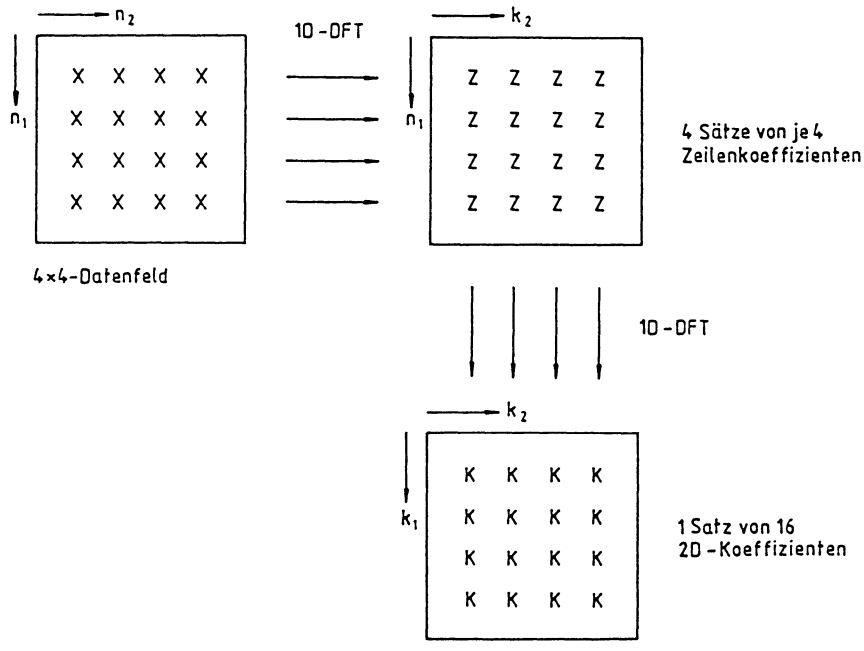


Bild 6.3. Durchführung des Zeile/Spalte-Algorithmus', nach [2.15]

Zur Berechnung durch zweimalige Anwendung der eindimensionalen Fourier-Transformation betrachte man zuerst die innere Summation in (6.6), die gegeben ist durch

$$X(n_1,k_2) = \sum_{n_2=0}^{N_2-1} x(n_1,n_2)\, W_{N2}^{k2n2} = x(n_1,0)+x(n_1,1)W_{N2}^{k2}+ \ldots +x(n_1,N_2-1)W_{N2}^{(N2-1)k2}. \tag{6.9}$$

Man beachte, daß die Gleichung (6.9) die diskrete Fourier-Transformierte der Spalten n_2 der Datenmatrix $[x(n_1,n_2)]$ ist. Man kann dann wie folgt weiter verfahren. Die Berechnung von (6.9) für alle Werte $n_1 = 0,1,\ldots,N_1-1$ ergibt eine neue Matrix

$$[X(k_1,n_2)] = \begin{bmatrix} X(0,0) & X(0,1) & \ldots & X(0,N_2-1) \\ X(1,0) & X(1,1) & \ldots & X(1,N_2-1) \\ X(2,0) & X(2,1) & \ldots & X(2,N_2-1) \\ . & \ldots & & . \\ . & \ldots & & . \\ . & \ldots & & . \\ X(N_1-1,0) & \ldots & \ldots & X(N_1-1,N_2-1) \end{bmatrix}.$$

Substitution von (6.9) in (6.7) führt zu

$$X_f(k_1,k_2) = \sum_{n_1=0}^{N_1-1} X(n_1,k_2)\, W_{N1}^{k1n1}. \tag{6.10}$$

Man kann also einen 1D-Algorithmus (z.B. den Cooley-Tukey-Algorithmus, vgl. Abschnitt 4.2.1) zweimal anwenden, um die 2D-FFT zu berechnen. Dabei ist jedoch zu beachten, daß die Anwendung von (6.10) auf Daten in einem Massenspeicher, der zeilenweise organisiert ist, eine Transponierung erfordert.

Matrixtransponieren und große I/O-Bandbreite sind zwei Hauptprobleme bei der Anwendung des Zeile/Spalte- (oder Spalte/Zeile-) Algorithmus'. Deshalb ist die Möglichkeit der Datenabspeicherung im Hauptspeicher entscheidend für eine schnelle Durchführung des Algorithmus'. Es ist anzustreben, das Matrixtransponieren zu vermeiden, und den Datenverkehr zwischen dem Hauptspeicher und dem Massenspeicher zu minimieren.

Für viele Anwendungsfälle ist die Datenmatrix in einem Massenspeicher abgespeichert, z.B. auf einer Festplatte oder auf einem Band, wo die leicht erreichbare kleinste Datenmenge eine vollständige Zeile ist. Diese Art der Datenabspeicherung erleichtert die Verarbeitung der eindimensionalen Daten, aber behindert die Verarbeitung von mehrdimensionalen Daten.

Konventionelle zweidimensionale FFT-Algorithmen werden sehr langsam, wenn die Matrixordnung so groß ist, daß sie nicht im Hauptspeicher gehalten werden kann. Eine derartige Situation erfordert dann, die Matrix zu transponieren. Die in den folgenden Abschnitten herzuleitenden Algorithmen wurden entworfen, um die Notwendigkeit des Transponierens zu beseitigen.

Eine Matrix auf einer Festplatte zu transponieren, hat nämlich manche Nachteile. Einerseits wird beträchtliche Zeit benötigt, um die Transposition auszuführen. Andererseits sind die Endergebnisse in transponierter Form. Dies bedingt im allgemeinen eine weitere Transponierung, um die Daten in Zeilenform für die Ausgabe umzusetzen.

6.2.2 Vektor-Radix-Algorithmus der DFT

Die genannten Nachteile des Zeile/Spalte-Algorithmus' haben zur Entwicklung von Methoden geführt, die direkt auf der 2D-Datenmenge operieren und auf diese Weise eine Transponierung von (Zwischen-)Ergebnissen entbehrlich machen. Darüberhinaus ergeben diese Algorithmen eine Einsparung im Rechenaufwand.

Die diskrete Berechnung der DFT aus der 2D-Datenmenge geht auf Arbeiten von G.E. Rivard [5.5] zurück. Diese in etwas ungewohnter mathematischer Darstellung gehaltenen Ergebnisse wurden später für allgemeine Radizes und nichtquadratische Datenarrays verallgemeinert [5.6] und mit dem Namen Vektor-Radix-Algorithmus (VR) belegt. Dabei deutet der Name darauf hin, daß für die verschiedenen Dimensionen unterschiedliche Radizes verwendet werden (können).

Im Gegensatz zum Zeile/Spalte-Algorithmus werden bei den direkten 2D-Algorithmen zweidimensionale Subarrays transformiert und deren 2D-Koeffizienten

rekursiv zu Koeffizienten höherer Ordnung verarbeitet. Dieses Prinzip wird hier zunächst allgemein dargestellt [5.6]. Dazu seien R_1 und R_2 Radizes derart, daß $N_1/R_1 = Q_1$ und $N_2/R_2 = Q_2$, wobei Q_1 und Q_2 natürliche Zahlen sind. Mit den Substitutionen

$$\begin{aligned} n_1 &= R_1 q_1 + u_1, & n_2 &= R_2 q_2 + u_2, \\ q_1 &= 0,1,\ldots,Q_1-1, & q_2 &= 0,1,\ldots,Q_2-1, \\ u_1 &= 0,1,\ldots,R_1-1, & u_2 &= 0,1,\ldots,R_2-1 \end{aligned}$$

folgt aus (6.6)

$$X_f(k_1,k_2) = \sum_{q_1=0}^{Q_1-1} \sum_{u_1=0}^{R_1-1} \sum_{q_2=0}^{Q_2-1} \sum_{u_2=0}^{R_2-1} x(R_1q_1+u_1,R_2q_2+u_2)\, W_{N1}^{k1(R1q1+u1)} W_{N2}^{k2(R2q2+u2)}.$$

Unter Beachtung von

$$W_{N1}^{k1(R1q1+u1)} = W_{Q1}^{k1q1}\, W_{N1}^{k1u1} \quad \text{und} \quad W_{N2}^{k2(R2q2+u2)} = W_{Q2}^{k2q2}\, W_{N2}^{k2u2}$$

erhält man

$$X_f(k_1,k_2) = \sum_{u_1=0}^{R_1-1} \sum_{u_2=0}^{R_2-1} W_{N1}^{k1u1}\, W_{N2}^{k2u2}\, X_{Q1Q2}^{u1u2}(k_1,k_2), \tag{6.11}$$

wobei

$$X_{Q1Q2}^{u1u2}(k_1,k_2) = \sum_{q_1=0}^{Q_1-1} \sum_{q_2=0}^{Q_2-1} x(R_1q_1+u_1,R_2q_2+u_2) W_{Q1}^{k1q1} W_{Q2}^{k2q2}. \tag{6.12}$$

Die Gleichungen (6.11) und (6.12) stellen die DFT eines $(Q_1 \times Q_2)$-Arrays dar.

Eine entsprechende Substitution für den Transformationsbereich mit

$$\begin{aligned} k_1 &= p_1 + v_1Q_1, & k_2 &= p_2 + v_2Q_2, \\ p_1 &= 0,1,\ldots,Q_1-1, & p_2 &= 0,1,\ldots,Q_2-1, \\ v_1 &= 0,1,\ldots,R_1-1, & v_2 &= 0,1,\ldots,R_2-1 \end{aligned}$$

ergibt

$$X(p_1+v_1Q_1,p_2+v_2Q_2) = \sum_{u_1=0}^{R_1-1} \sum_{u_2=0}^{R_2-1} W_{N1}^{(p1+v1Q1)u1} W_{N2}^{(p2+v2Q2)u2} X_{Q1Q2}^{u1u2}(p_1+v_1Q_1,p_2+v_2Q_2).$$

Ferner gilt

$$W_{N1}^{(p1+v1Q1)u1} = W_{N1}^{p1u1}\, W_{R1}^{v1u1} \qquad \text{und} \qquad W_{N2}^{(p2+v2Q2)u2} = W_{N2}^{p2u2}\, W_{R2}^{v2u2}$$

sowie wegen der Periodizität der $(Q_1 \times Q_2)$-Punkte-DFT:

$$X_{Q1Q2}^{u1u2}(p_1+v_1Q_1, p_2+v_2Q_2) = X_{Q1Q2}^{u1u2}(p_1, p_2).$$

Mit diesen Gleichungen folgt die allgemein gültige Beziehung für einen Schmetterling:

$$X(p_1+v_1Q_1, p_2+v_2Q_2) = \sum_{u_1=0}^{R_1-1} \sum_{u_2=0}^{R_2-1} W_{R1}^{v1u1}\, W_{R2}^{v2u2}\, \{W_{N1}^{p1u1}\, W_{N2}^{p2u2}\, X_{Q1Q2}^{u1u2}(p_1, p_2)\}. \qquad (6.13)$$

Die Interpretation dieses Ausdrucks zeigt, daß es insgesamt Q_1Q_2 Schmetterlinge gibt, die R_1R_2 Werte berechnen. Die Eingangswerte werden mit den in eckigen Klammern angegebenen Drehfaktoren vormultipliziert. Die geschweiften Klammern bezeichnen die 2D-DFT der Ordnung R_1R_2. Das Produkt der Drehfaktoren (in geschweiften Klammern) kann vorab berechnet und abgespeichert werden, was einen bedeutenden rechentechnischen Vorteil bietet.

Ein Grundschmetterling für den Fall $R_1 = R_2 = 2$ ist in Bild 6.4 dargestellt. Für diesen Fall reduziert sich (6.13) auf

$$X(p_1+v_1(N/2), p_2+v_2(N/2)) = \sum_{u_1=0}^{1} \sum_{u_2=0}^{1} (-1)^{v1u1+v2u2}\, \{W_N^{p1u1+p2u2}\, X_{(N/2)(N/2)}^{u1u2}(p_1, p_2)\}$$

mit $p_1, p_2 = 0,1,\ldots,(N/2)-1$ und $v_1, v_2 = 0,1$.

Die Struktur im Bild 6.4 hat eine minimale Anzahl der Operationen. Der zugehörige Flußgraph des Algorithmus' ist im Bild 6.5 dargestellt. Tabelle 6.1 gibt die Anzahl der Multiplikationen und der Plattenzugriffe (falls erforderlich) für die Vektor-Radix-Methode (VR) und die Zeile/Spalte-Zerlegung (ZS) für verschiedene Radizes und $N=2^r$ an.

Für die weitere Darstellung der diskreten 2D-DFT folgen wir [5.7] und [5.8]. Wir erinnern zunächst an die Definition der zweidimensionalen DFT

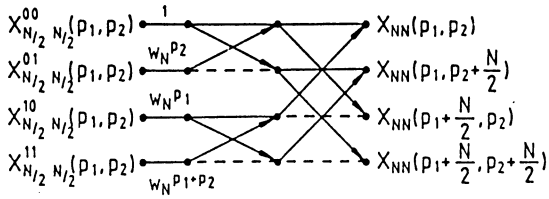


Bild 6.4. Schmetterling für den Vektor-Radix-Algorithmus, nach [5.6]

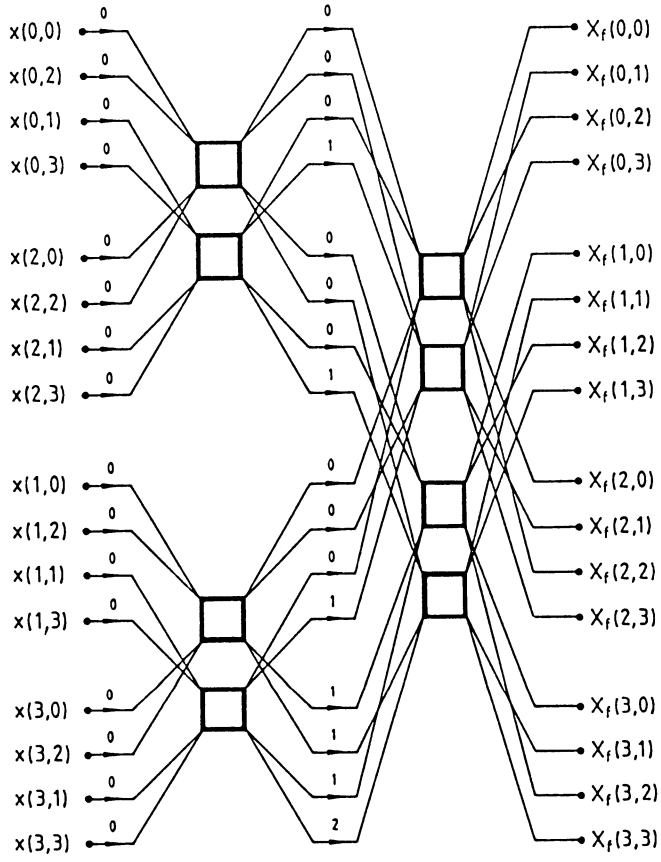


Bild 6.5. SFG einer (4×4)-VR-FFT. Die Ziffern bezeichnen die Exponenten der Drehfaktoren, nach [5.6]

Tabelle 6.1. Anzahl der Multiplikationen für verschiedene (N×N)-DFT-Algorithmen ($N=2^r$) [5.6]

FFT-Typ	Radix	Plattenzugriffe			
		2	4	8	16
VR	2×2	r+1	(r/2)+1	(r/3)+1	(r/4)+1
VR	4×4		(r/2)+1	(r/3)+1	(r/4)+1
VR	8×8			(r/3)+1	(r/4)+1
RC	2	r	r/2	r/3	r/4
RC	4	r	r/2	r/3	r/4
RC	8	r	r/2	r/3	r/4
RC	16	r	r/2	r/3	r/4

FFT-Typ	Radix	Multiplikationen je Schmetterling	Anzahl der Schmetterlinge
VR	2×2	3+0	$(1/4)N^2r$
VR	4×4	15+0	$(1/32)N^2r$
VR	8×8	63+24	$(1/192)N^2r$
RC	2	1+0	N^2r
RC	4	3+0	$(1/4)N^2r$
RC	8	7+2	$(1/12)N^2r$
RC	16	15+9	$(1/32)N^2r$

FFT-Typ	Radix	Gesamtzahl der Multiplikationen	Verhältnis
VR	2×2	$(3/4)N^2(r-1)$	0,75
VR	4×4	$(15/32)N^2(r-2)$	0,426
VR	8×8	$(29/64)N^2(r-(63/29))$	0,405
RC	2	$N^2(r-1)$	1,00
RC	4	$(3/4)N^2(r-2)$	0,682
RC	8	$(3/4)N^2(r-(7/3))$	0,659
RC	16	$(3/4)N^2(r-(5/2))$	0,648

$$X_f(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1,n_2) W_{N1}^{n1k1} W_{N2}^{n2k2}.$$

Im folgenden beschränken wir uns auf die Betrachtung von Radix-2-Algorithmen. Weil N_1 und N_2 dann Potenzen von 2 sind, kann jede der Summen in zwei Teile mit der gleichen Anzahl von Gliedern aufgeteilt werden. Die zweidimensionale DFT wird nun in Matrixform geschrieben. Die Matrix $[x(n_1,n_2)]$ kann

in vier Untermatrizen $[a(n_1,n_2)]$, $[b(n_1,n_2)]$, $[c(n_1,n_2)]$ und $[d(n_1,n_2)]$ zerlegt werden, d.h.

$$[x(n_1,n_2)] = \begin{bmatrix} [a(n_1,n_2)] & [b(n_1,n_2)] \\ [c(n_1,n_2)] & [d(n_1,n_2)] \end{bmatrix}.$$

Dann haben wir

$$x(n_1,n_2) = \begin{cases} a(n_1,n_2) & 0 \leq n_1,n_2 \leq (N/2)-1 \\ b(n_1,n_2-(N/2)) & 0 \leq n_1 \leq (N/2)-1, \text{ und } (N/2) \leq n_2 \leq N-1, \\ c(n_1-(N/2),n_2) & 0 \leq n_2 \leq (N/2)-1, \text{ und } (N/2) \leq n_1 \leq N-1, \\ d(n_1-(N/2),n_2-(N/2)) & (N/2) \leq n_1,n_2 \leq N-1. \end{cases}$$

Mit dieser Definition erhält man

$$X_f(k_1,k_2) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1,n_2)\, W_N^{n_1k_1}\, W_N^{n_2k_2}$$

$$= \sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} \{ a(n_1,n_2) + b(n_1,n_2)\exp(-j\pi k_2) + c(n_1,n_2)\exp(-j\pi k_1)$$

$$+ d(n_1,n_2)\exp(-j\pi(k_1+k_2)) \}\, W_N^{n_1k_1}\, W_N^{n_2k_2}.$$

Die Koeffizientenmatrix wird ebenfalls in vier Untermatrizen aufgeteilt: $[X_f(2k_1,2k_2)]$, $[X_f(2k_1,2k_2+1)]$, $[X_f(2k_1+1,2k_2)]$ und $[X_f(2k_1+1,2k_2+1)]$. Mit diesen Untermatrizen stellt sich die Transformation wie folgt dar:

$$A(k_1,k_2) = X_f(2k_1,2k_2) =$$

$$\sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} \{a(n_1,n_2)+b(n_1,n_2)+c(n_1,n_2)+d(n_1,n_2)\} W_{N/2}^{n_1k_1} W_{N/2}^{n_2k_2},$$

$$B(k_1,k_2) = X_f(2k_1,2k_2+1) =$$

$$\sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} \{a(n_1,n_2)-b(n_1,n_2)+c(n_1,n_2)-d(n_1,n_2)\} W_{N/2}^{n_1k_1} W_{N/2}^{n_2k_2},$$

$$C(k_1,k_2) = X_f(2k_1+1,2k_2) =$$

$$\sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} \{a(n_1,n_2)+b(n_1,n_2)-c(n_1,n_2)-d(n_1,n_2)\} W_{N/2}^{n_1 k_1} W_{N/2}^{n_2 k_2},$$

$$D(k_1,k_2) = X_f(2k_1+1,2k_2+1) =$$

$$\sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} \{a(n_1,n_2)-b(n_1,n_2)-c(n_1,n_2)+d(n_1,n_2)\} W_{N/2}^{n_1 k_1} W_{N/2}^{n_2 k_2},$$

mit $k_1, k_2 = 0,1,\ldots,(N/2)-1.$ (6.14)

Diese Gleichungen werden rekursiv auf die Untermatrizen angewandt, bis man schließlich die Transformierten von (2×2)-Untermatrizen erhält.

Eine Betrachtung des Bildes 6.6 zeigt, daß die vereinfachte Schmetterlings-Einheit vier Eingaben a, b, c, d braucht, um daraus durch vier Transformationen gemäß (6.14) (d.h. gerade-gerade, gerade-ungerade, ungerade-gerade und ungerade-ungerade) die vier Ausgaben A, B, C und D zu erzeugen.

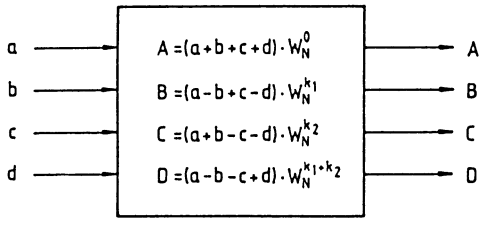


Bild 6.6. Vereinfachte Schmetterlingseinheit für den VR-Algorithmus, nach [5.6]

6.2.3 Direkter 2D-FFT-Algorithmus

Zur Veranschaulichung der direkten 2D-FFT referieren wir hier noch eine alternative Herleitung [5.7].

Sei $[x(n_1,n_2)]$ eine (N×N)-Matrix, $(N=2M=2^r)$, dann kann ihre Transformierte für $k_1,k_2 = 1,2,\ldots,N-1$ wie folgt zerlegt werden:

$$X_f(k_1,k_2) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1,n_2)\exp(-j2\pi(n_1k_1+n_2k_2)/N)$$

$$
\begin{aligned}
&= \sum_{n_1=0}^{2M-1} \Big\{ \sum_{n_2=0}^{M-1} x(n_1,2n_2)\exp(-j2\pi(n_1k_1+2n_2k_2)/(2M)) \\
&+ \sum_{n_2=0}^{M-1} x(n_1,2n_2+1)\exp(-2\pi j(k_1n_1+2k_2n_2+k_2)/(2M)) \Big\} \\
&= \sum_{n_1=0}^{M-1} \Big\{ \sum_{n_2=0}^{M-1} x(2n_1,2n_2)\exp(-2\pi j(2k_1n_1+2k_2n_2)/(2M)) \\
&+ \sum_{n_2=0}^{M-1} x(2n_1,2n_2+1)\exp(-2\pi j(2k_1n_1+2k_2n_2+k_2)/(2M)) \Big\} \\
&+ \sum_{n_1=0}^{M-1} \Big\{ \sum_{n_2=0}^{M-1} x(2n_1+1,2n_2)\exp(-2\pi j(2k_1n_1+2k_2n_2+k_1)/(2M)) \\
&+ \sum_{n_2=0}^{M-1} x(2n_1+1,2n_2+1)\exp(-2\pi j(2n_1k_1+2n_2k_2+k_1+k_2)/(2M)) \Big\}. \qquad (6.15)
\end{aligned}
$$

Zur Vereinfachung definiert man die folgenden Ausdrücke

$$
\begin{aligned}
q_1(n_1,n_2) &= x(2n_1,2n_2) \\
q_2(n_1,n_2) &= x(2n_1,2n_2+1) \\
q_3(n_1,n_2) &= x(2n_1+1,2n_2) \\
q_4(n_1,n_2) &= x(2n_1+1,2n_2+1) \\
n_1 &= 0,1,\ldots,N-1, \qquad n_2 = 0,1,\ldots,N-1. \qquad (6.16)
\end{aligned}
$$

Damit folgt aus (6.15)

$$
\begin{aligned}
X_f(k_1,k_2) &= \sum_{n_1=0}^{M-1} \sum_{n_2=0}^{M-1} q_1(n_1,n_2)\exp(-2\pi j(k_1n_1+k_2n_2)/(2M)) \\
&+ \sum_{n_1=0}^{M-1} \sum_{n_2=0}^{M-1} q_2(n_1,n_2)\exp(-2\pi j(2k_1n_1+2k_2n_2+k_2)/(2M)) \\
&+ \sum_{n_1=0}^{M-1} \sum_{n_2=0}^{M-1} q_3(n_1,n_2)\exp(-2\pi j(2n_1k_1+2n_2k_2+k_1)/(2M)) \\
&+ \sum_{n_1=0}^{M-1} \sum_{n_2=0}^{M-1} q_4(n_1,n_2)\exp(-2\pi j(2n_1k_1+2n_2k_2+k_1+k_2)/(2M)).
\end{aligned}
$$

Mit

$$Q_i(k_1,k_2) = \sum_{n_1=0}^{M-1} \sum_{n_2=0}^{M-1} q_i(n_1,n_2)\exp(-2\pi j(n_1k_1+n_2k_2)/M) \qquad i = 1,2,3,4$$

ergibt sich

$$X_f(k_1,k_2) = Q_1(k_1,k_2) + \exp(-\pi jk_2/M)\cdot Q_2(k_1,k_2) + \exp(-\pi jk_1/M)\cdot Q_3(k_1,k_2) + \exp(-\pi j(k_1+k_2)/M)\cdot Q_4(k_1,k_2). \qquad (6.17)$$

In (6.17) müssen die Untermatrizen $[Q_1]$, $[Q_2]$, $[Q_3]$ und $[Q_4]$ auf k_1, k_2 = 0,1,....2N-1 vergrößert werden. Dieser Bereich kann wegen der Periodizität der DFT auf jeden Fall benutzt werden, z.B.

$$Q_i(k_1,k_2) = Q_i(k_1+M,k_2) = Q_i(k_1,k_2+M) = Q_i(k_1+M,k_2+M) , \qquad i = 1,2,3,4.$$

Um die Fourier-Transformierte $X_f(k_1,k_2)$ zu erhalten, wird (6.17) rekursiv r-mal angewandt. Zunächst müssen die Elemente von [x] derart neu geordnet werden, daß die umgeordnete Matrix wie folgt aufgeteilt werden kann

$$\left[\begin{array}{c|c} [q_1] & [q_2] \\ \hline [q_3] & [q_4] \end{array}\right].$$

Dazu wurden q_1, q_2, q_3 und q_4 in (6.16) definiert.

Das Problem ist nun darauf reduziert, $Q_1(k_1,k_2)$, $Q_2(k_1,k_2)$, $Q_3(k_1,k_2)$ und $Q_4(k_1,k_2)$ zu bestimmen. Jede dieser Untermatrizen kann durch Verwendung von (6.17) ermittelt werden. Benutzt man dieses Verfahren der Aufteilung über r Stufen, so hat man am Ende 4^{r-1} (2x2)-Untermatrizen und kann (6.17) auf jede (2x2)-Untermatrix anwenden. Im weiteren benutzt man (6.17) zur Gewinnung der Transformierten von (4x4)-, (8x8)-, (16x16)-,... -Blöcken, bis man schließlich $X_f(k_1,k_2)$ erhält.

Mit $W_N = \exp(-2\pi j/N)$ kann (6.17) vereinfacht werden zu

$$X_f(k_1,k_2) = Q_1(k_1,k_2) + W_N^{k2}\cdot Q_2(k_1,k_2) + W_N^{k1}\cdot Q_3(k_1,k_2) + W_N^{k1+k2}\cdot Q_4(k_1,k_2). \qquad (6.18)$$

Die Untermatrizen Q_1, Q_2, Q_3 und Q_4 wurden nur für k_1, $k_2 = 0,1,\ldots,(N/2)-1$ definiert. Wegen ihrer Periodizität kann man die Bereiche wie folgt erweitern:

$$\left[\begin{array}{c|c} \text{Quadrant 1} & \text{Quadrant 2} \\ [k_1,k_2] & [k_1,k_2+(N/2)] \\ \hline \text{Quadrant 3} & \text{Quadrant 4} \\ [k_1+(N/2),k_2] & [k_1+(N/2),k_2+(N/2)] \end{array}\right].$$

Das oben erwähnte Umordnungsverfahren bewirkt, daß die Elemente aus [x] an der Position (i,j) (i,j = 0,1,...,N-1) zur Position (i',j') verschoben werden. Im ersten Schritt wandelt man die Indizes i und j in ihre Binärdarstellung. Anschließend wendet man die Bitumkehroperation (BRO) an. Im letzten Schritt führt man sie zurück in die Dezimaldarstellung. Danach hat man die neue Adresse (i',j'). Beispiel:

$$[x] = \begin{bmatrix} x(0,0) & x(0,1) & x(0,2) & x(0,3) \\ x(1,0) & x(1,1) & x(1,2) & x(1,3) \\ x(2,0) & x(2,1) & x(2,2) & x(2,3) \\ x(3,0) & x(3,1) & x(3,2) & x(3,3) \end{bmatrix}$$

$\downarrow$ Konvertierung ins Binärsystem

$$= \begin{bmatrix} x(00,00) & x(00,01) & x(00,10) & x(00,11) \\ x(01,00) & x(01,01) & x(01,10) & x(01,11) \\ x(10,00) & x(10,01) & x(10,10) & x(10,11) \\ x(11,00) & x(11,01) & x(11,10) & x(11,11) \end{bmatrix}$$

$\downarrow$ BRO und Konvertierung ins Dezimalsystem

$$[x'] = \begin{bmatrix} x(0,0) & x(0,2) & x(0,1) & x(0,3) \\ x(2,0) & x(2,2) & x(2,1) & x(2,3) \\ x(1,0) & x(1,2) & x(1,1) & x(1,3) \\ x(3,0) & x(3,2) & x(3,1) & x(3,3) \end{bmatrix}.$$

Man beachte, daß der Transformationskern W_N die folgenden Eigenschaften hat:

$$(1)\ W_N^{k1+(N/2)} = W_N^{k1} \cdot W_N^{N/2} = -W_N^{k1},$$

$$(2)\ W_N^{k2+(N/2)} = -W_N^{k2},$$

$$(3)\ W_N^{k1+(N/2)+k2+(N/2)} = W_N^{(k1+k2)+N} = W_N^N \cdot W_N^{k1+k2} = W_N^{k1+k2}. \qquad (6.19)$$

Die Eigenschaften in (6.19) werden zur Erzeugung einer vereinfachten Version von (6.18) auf jeden der vier Quadranten angewandt. Diese Quadrantengleichungen sind in Tabelle 6.2 angegeben. Sie werden rekursiv von dem noch zu beschreibenden Algorithmus zur Gewinnung der zweidimensionalen FFT verwendet.

Tabelle 6.2. Rekursive Gleichungen zur Erzeugung der FFT einer (N×N)-Matrix mit M=N/2 [5.7]

Quadrant	Verwendete Gleichungen
1	$X_f(k_1,k_2) = Q_1(k_1,k_2) + W_N^{k2}\, Q_2(k_1,k_2) + W_N^{k1}\, Q_3(k_1,k_2) + W_N^{k1+k2}\, Q_4(k_1,k_2)$
2	$X_f(k_1,k_2+M) = Q_1(k_1,k_2+M) + W_N^{k2+M}\, Q_2(k_1,k_2+M) + W_N^{k1}\, Q_3(k_1,k_2+M) + W_N^{k1+k2+M}\, Q_4(k_1,k_2+M) = Q_1(k_1,k_2) - W_N^{k2}\, Q_2(k_1,k_2) + W_N^{k1}\, Q_3(k_1,k_2) - W_N^{k1+k2}\, Q_4(k_1,k_2)$
3	$X_f(k_1+M,k_2) = Q_1(k_1+M,k_2) + W_N^{k2}\, Q_2(k_1+M,k_2) + W_N^{k1+M}\, Q_3(k_1+M,k_2) + W_N^{k1+k2+M}\, Q_4(k_1+M,k_2) = Q_1(k_1,k_2) + W_N^{k2}\, Q_2(k_1,k_2) - W_N^{k1}\, Q_3(k_1,k_2) - W_N^{k1+k2}\, Q_4(k_1,k_2)$
4	$X_f(k_1+M,k_2+M) = Q_1(k_1+M,k_2+M) + W_N^{k2+M}\, Q_2(k_1+M,k_2+M) + W_N^{k1+M}\, Q_3(k_1+M,k_2+M) + W_N^{k1+k2+2M}\, Q_4(k_1+M,k_2+M) = Q_1(k_1,k_2) - W_N^{k2}\, Q_2(k_1,k_2) - W_N^{k1}\, Q_3(k_1,k_2) + W_N^{k1+k2}\, Q_4(k_1,k_2)$

Man braucht also die Größen

$$Q_1(k_1,k_2),\quad W_N^{k2}\cdot Q_2(k_1,k_2),\quad W_N^{k1}\cdot Q_3(k_1,k_2)\quad \text{und}\quad W_N^{k1+k2}\cdot Q_4(k_1,k_2)$$

jeder Stufe lediglich einmal zu bestimmen. Die anderen drei Quadrantengleichungen können durch die entsprechenden in Tabelle 6.2 gezeigten Additionen und Subtraktionen gefunden werden. Der Algorithmus findet die FFT indem die Quadrantengleichungen zur Transformation aller (2x2)-Untermatrizen verwendet werden. Die Quadrantengleichungen werden dann auf diese (2x2)-Ergebnismatrizen angewandt, und man erhält (4x4)-Untermatrizen. Dieses Verfahren setzt man bis zur letzten Stufe fort.

Für eine (NxN)-Matrix ($N=2^r$) sind folgende Rechenschritte erforderlich:

(1) Man ordnet die Elemente nach der BEO(1) (siehe Abschnitt 3.4.3) um.

(2) Man benutzt die vier Quadrantengleichungen zur Gewinnung der Fourier-Transformierten von $2^{r-1}x2^{r-1} = 4^{r-1}$ (2x2)-Untermatrizen. (Bemerkung: N=2).

(3) Für p=2,....,r benutzt man iterativ die Quadrantengleichungen zur Erzeugung der Fourier-Transformierten von 4^{r-1} ($2^p x 2^p$)-Untermatrizen, die in der Stufe (p-1) transformiert wurden. (Bemerkung: $N=2^p$).

(4) Wenn p=r, erhält man $X_f(k_1,k_2)$.

Beispiel [5.7]:

(1)
$$[x(n_1,n_2)] = \begin{bmatrix} 1 & 5 & 2 & 6 \\ 9 & 13 & 10 & 1 \\ 3 & 7 & 4 & 8 \\ 11 & 15 & 12 & 16 \end{bmatrix} \xrightarrow{\text{BRO}} \begin{bmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \\ 9 & 10 & 13 & 14 \\ 11 & 12 & 15 & 16 \end{bmatrix},$$

d.h.

$$[q_1] = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix},\ [q_2] = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix},\ [q_3] = \begin{bmatrix} 9 & 10 \\ 11 & 12 \end{bmatrix},\ [q_4] = \begin{bmatrix} 13 & 14 \\ 15 & 16 \end{bmatrix}.$$

(2) Wir benutzen nun die Quadrantengleichungen zur Gewinnung der Fourier-Transformierten der (2×2)-Untermatrizen

$$[x_1] = (1/2)\left[\begin{array}{cc|cc} 1+2+3+4 & 1-2+6-2 & 5+6+7+8 & 5-6+7-8 \\ 1+2-6-2 & 1-2-3+4 & 5+6-7-8 & 5-6-7+8 \\ \hline 9+10+11+12 & 9-10+11-12 & 13+14+15+16 & 13-14+15-16 \\ 9+10-11-12 & 9-10-11+12 & 13+14-15-16 & 13-14-15+16 \end{array}\right]$$

$$= \left[\begin{array}{cc|cc} 5 & -1 & 13 & -1 \\ -2 & 0 & -2 & 0 \\ \hline 21 & -1 & 29 & -1 \\ -2 & 0 & -2 & 0 \end{array}\right].$$

(3) Nun wenden wir die Quadrantengleichungen iterativ auf die transformierten (2×2)-Untermatrizen zur Gewinnung der Fourier-Transformierten der (4×4)-Untermatrizen an:

$$[Q_1] = T\{[q_1]\} = \begin{bmatrix} 5 & -1 \\ -2 & 0 \end{bmatrix}, \quad [Q_2] = T\{[q_2]\} = \begin{bmatrix} 13 & -1 \\ -2 & 0 \end{bmatrix},$$

$$[Q_3] = T\{[q_3]\} = \begin{bmatrix} 21 & -1 \\ -2 & 0 \end{bmatrix}, \quad [Q_4] = T\{[q_4]\} = \begin{bmatrix} 29 & -1 \\ -2 & 0 \end{bmatrix},$$

$$[W_4^{k2}Q_2] = \begin{bmatrix} 13 & j \\ -2 & 0 \end{bmatrix}, \quad [W_4^{k1}Q_3] = \begin{bmatrix} 21 & -1 \\ 2j & 0 \end{bmatrix}, \quad [W_4^{k1+k2}Q_4] = \begin{bmatrix} 29 & j \\ 2j & 0 \end{bmatrix}.$$

(4) Aus (6.17) erhalten wir:

$$[X_f(k_1,k_2)] = \left[\begin{array}{c|c} \text{Quadrant 1} & \text{Quadrant 2} \\ \hline \text{Quadrant 3} & \text{Quadrant 4} \end{array}\right].$$

$$\text{Quadrant 1} : [Q_1 + W_4^{k2}\cdot Q_2 + W_4^{k1}\cdot Q_3 + W_4^{k1+k2}\cdot Q_4] = \begin{bmatrix} 68 & -2+2j \\ -4+4j & 0 \end{bmatrix}.$$

$$\text{Quadrant 2} : [Q_1 - W_4^{k2}\cdot Q_2 + W_4^{k1}\cdot Q_3 - W_4^{k1+k2}\cdot Q_4] = \begin{bmatrix} -16 & -2+2j \\ 0 & 0 \end{bmatrix}.$$

$$\text{Quadrant 3} : [Q_1 + W_4^{k2}\cdot Q_2 - W_4^{k1}\cdot Q_3 - W_4^{k1+k2}\cdot Q_4] = \begin{bmatrix} -32 & 0 \\ -4-4j & 0 \end{bmatrix}.$$

$$\text{Quadrant 4} : [Q_1 - W_4^{k2}\cdot Q_2 - W_4^{k1}\cdot Q_3 - W_4^{k1+k2}\cdot Q_4] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

(5) Die vollständige Koeffizientenmatrix ist somit

$$[X_f(k_1,k_2)] = \left[\begin{array}{cc|cc} 68 & -2+2j & -16 & -2-2j \\ -4+4j & 0 & 0 & 0 \\ \hline -32 & 0 & 0 & 0 \\ -4-4j & 0 & 0 & 0 \end{array}\right].$$

Damit ist der Algorithmus mit p=2 durchgeführt, d.h. für dieses Beispiel ($N=2^2$) die 2D-FFT bestimmt.

Die Tabelle 6.3 gibt die Anzahl der Additionen und Multiplikationen an, die in jeder Stufe zur Ausführung des Algorithmus' benötigt werden. Für Stufe i ist die Anzahl der Additionen und Multiplikationen zur Transformation aller ($2^i x 2^i$)-Untermatrizen angegeben. Weil der Algorithmus nach der r-ten Stufe endet, ist die Anzahl der Operationen (entsprechend Tabelle 6.3):

$$\sum_{i=1}^{r} (3\cdot 4^r + 3\cdot 4^i) = 3\cdot r\cdot 4^r + 3\sum_{i=1}^{r} 4^i = 3\cdot r\cdot 4^r + 3\cdot 4(4^r-1)/3$$

$$= 3\cdot r\cdot 4^r + 4(4^r-1) = 3\cdot r\cdot 4^r + 4\cdot 4^r - 4$$

$$= 3(2^r)^2\log_2 N + 4(2^r)^2 - 4 = 3N^2\log_2 N + 4N^2 - 4.$$

Tabelle 6.3. Anzahl der Additionen und Multiplikationen für die direkte zweidimensionale FFT [5.7]

Stufe	Matrixgröße	Additionen	Multiplikationen
1	2 × 2	3 Add. je Matrix. Es gibt $2^{r-1} \cdot 2^{r-1} = 4^{r-1}$ Matrizen, d.h. Anzahl der Additionen : $3 \cdot 4 \cdot 4^{r-1} = 3 \cdot 4^r$	0
2	4 × 4	3 Add. je Matrix. Es gibt $2^{r-2} \cdot 2^{r-2} = 4^{r-2}$ Matrizen, d.h. Anzahl der Additionen : $3 \cdot 4^2 \cdot 4^{r-2} = 3 \cdot 4^r$	3 Multiplikationen für 16 Elemente $3 \cdot 4^2$
3	8 × 8	3 Add. je Matrix. Es gibt $2^{r-3} \cdot 2^{r-3} = 4^{r-3}$ Matrizen, d.h. Anzahl der Additionen : $3 \cdot 4^3 \cdot 4^{r-3} = 3 \cdot 4^r$	3 Multiplikationen für 64 Elemente $3 \cdot 4^3$

Die Standardmethode der FFT (Zeile/Spalte-Algorithmus) benötigt $4N^2 \log_2 N$ Additionen und Multiplikationen. Damit folgt

$$3N^2 \log_2 N + 4N^2 - 4 < 4N^2 \log_2 N \qquad \text{für } N \geq 16.$$

Tabelle 6.4 gibt einen Vergleich der beiden Methoden für verschiedene N. Die wesentlichen Vorteile dieser Methode liegen auf folgenden Gebieten: Da keine Transponierung erforderlich wird, ist diese Methode effizient zur Bestimmung der Fourier-Transformierten einer Matrix, die nicht im Hauptspeicher Platz findet. Durch Aufteilung in mehrere Untermatrizen und durch das Arbeiten mit kleineren Untermatrizen ist die Methode außerdem besonders für die Verarbeitung durch ein Multiprozessorsystem geeignet.

Tabelle 6.4. Vergleich der Anzahl der Multiplikationen für verschiedene Werte von N [5.7]

N x N	direkte Methode	Zeile/Spalte-Methode
4 x 4	156	128
8 x 8	1.084	768
16 x 16	4.092	4.096
32 x 32	19.452	20.480
512 x 512	8.126.460	9.437.184
1024 x 1024	35.651.580	41.943.040

6.2.4 Berechnung der 2D-FFT aus Koeffizienten niedrigerer Ordnung

Im Abschnitt 6.1 haben wir gezeigt, daß die zweidimensionale DWHT in hierarchischer Weise ausgeführt werden kann. In diesem Abschnitt untersuchen wir, ob die zweidimensionale DFT ebenfalls hierarchisch ausgeführt werden kann. Wie sich zeigen wird, ist eine strenge hierarchische Entwicklung nicht möglich, wohl aber ein Koeffizientenaufbau aus jeweils einem Koeffizienten von 4 Quadranten.

Die Matrixdarstellung für die zweidimensionale (N×N)-DFT ($N=2^r$) ist (vgl. Abschnitt 5.1)

$$[X_f] = [W_N^E][x][W_N^E]. \tag{6.20}$$

Die zweidimensionale (N×N)-DWHT kann wie folgt dargestellt werden:

$$[X_{WH}] = [H_H(L)][x][H_H(L)]^T, \qquad L = 0,1,\ldots,r, \qquad N=2^r. \tag{6.21}$$

Mit (6.20) und (6.21) folgt

$$[X_f] = [T(L)][X_{WH}][T(L)]^T,$$

wobei $[T(L)] = [W_N^E][H_H(L)]$ eine (N×N)-Transformationsmatrix ist.

Wie im Abschnitt 4.2.6 zerlegen wir jetzt die Matrix $[T_k]$ in N Zeilenvektoren und wenden die Bitumkehroperationen auf diese Matrix $[T_k]$ mit Zeilenvektoren an. Die so erhaltene Konversionsmatrix $[T_{<k>}(L)]$ ist orthogonal und hat eine blockdiagonale Struktur

$$[X_{f<k>}(L)] = [T_{<k>}(L)][X_{WH}(L)][T_{<k>}(L)]^T. \tag{6.22}$$

Die zweidimensionale DWHT kann in folgenden Schritten hierarchisch ausgeführt werden [3.1]:

$[H_H(L)]$ kann auf $[H_H(L-1)]$ zurückgeführt werden und ein (L×L)-Datenblock $[X(L)]$ läßt sich in vier Quadrantenmatrizen $[Q_{ij}(L-1)]$ (i,j = 1,2) zerlegen. Damit wird

$$[X_{WH}(L)] =$$

$$\begin{bmatrix} [H_H(L-1)] & [H_H(L-1)] \\ [H_H(L-1)] & -[H_H(L-1)] \end{bmatrix} \begin{bmatrix} [Q_{11}(L-1)] & [Q_{12}(L-1)] \\ [Q_{21}(L-1)] & [Q_{22}(L-1)] \end{bmatrix} \begin{bmatrix} [H_H(L-1)] & [H_H(L-1)] \\ [H_H(L-1)] & -[H_H(L-1)] \end{bmatrix}^T.$$

Wir setzen ferner

$$[X_{WH}(L)] = \begin{bmatrix} [U_{11}] & [U_{12}] \\ [U_{21}] & [U_{22}] \end{bmatrix}. \tag{6.23}$$

wobei $[U_{11}] = [X_{WH11}(L-1)] + [X_{WH12}(L-1)] + [X_{WH21}(L-1)] + [X_{WH22}(L-1)]$,
$[U_{12}] = [X_{WH11}(L-1)] - [X_{WH12}(L-1)] + [X_{WH21}(L-1)] - [X_{WH22}(L-1)]$,
$[U_{21}] = [X_{WH11}(L-1)] + [X_{WH12}(L-1)] - [X_{WH21}(L-1)] - [X_{WH22}(L-1)]$,
$[U_{22}] = [X_{WH11}(L-1)] - [X_{WH12}(L-1)] - [X_{WH21}(L-1)] + [X_{WH22}(L-1)]$.

Im Abschnitt 4.2.6 haben wir [T(L)] wie folgt berechnet

$$[T_{<k>}(L)] = \begin{bmatrix} [T_{<k>}(L-1)] & 0 \\ 0 & [R_{<k>}(L-1)] \end{bmatrix}. \tag{6.24}$$

wobei $[T_{<k>}(L)]$ die Konventierungsmatrix und $[R_{<k>}(L)]$ eine Restmatrix ist.

Nun setzen wir (6.23) und (6.24) in (6.22) ein und erhalten

$$[X_{f<k>}(L)] = \begin{bmatrix} [T_{<k>}(L-1)][U_{11}][T_{<k>}(L-1)]^T & [T_{<k>}(L-1)][U_{12}][R(L-1)]^T \\ [R_{<k>}(L-1)][U_{21}][T_{<k>}(L-1)]^T & [R_{<k>}(L-1)][U_{22}][R(L-1)]^T \end{bmatrix}.$$

Wir führen nun die folgenden Abkürzungen ein

$$[X_{f<k>}(L)] = \begin{bmatrix} P'_{11} & P'_{12} \\ P'_{21} & P'_{22} \end{bmatrix} \tag{6.25}$$

und $[Z_{<k>}(L-1)] = [R_{<k>}(L-1)][T_{<k>}(L-1)]^T.$ (6.26)

Dabei sind die $[P'_{ij}]$ wie folgt definiert:

$$[P'_{11}] = [P_{11}] = [T_{<k>}(L-1)][U_{11}][T_{<k>}(L-1)]^T.$$

$$\begin{aligned}[P'_{12}] &= [T_{<k>}(L-1)][U_{12}][R_{<k>}(L-1)]^T \\ &= [T_{<k>}(L-1)][U_{12}][T_{<k>}(L-1)]^T[T_{<k>}(L-1)][R_{<k>}(L-1)]^T \\ &= [P_{12}][T_{<k>}(L-1)][R_{<k>}(L-1)]^T = [P_{12}][Z_{<k>}(L-1)]^T\end{aligned}$$

(Bemerkung : $[P_{12}] = [T_{<k>}(L-1)][U_{12}][T_{<k>}(L-1)]^T$).

$$\begin{aligned}[P'_{21}] &= [R_{<k>}(L-1)][U_{21}][T_{<k>}(L-1)]^T \\ &= [R_{<k>}(L-1)][T_{<k>}(L-1)]^T[T_{<k>}(L-1)][U_{21}][T_{<k>}(L-1)]^T \\ &= [Z_{<k>}(L-1)][P_{21}]\end{aligned}$$

(Bemerkung : $[P_{21}] = [T_{<k>}(L-1)][U_{21}][T_{<k>}(L-1)]^T$).

$$\begin{aligned}[P'_{22}] &= [R_{<k>}(L-1)][U_{22}][R_{<k>}(L-1)]^T \\ &= [Z_{<k>}(L-1)][P_{22}][Z_{<k>}(L-1)]^T\end{aligned}$$

(Bemerkung : $[P_{22}] = [T_{<k>}(L-1)][U_{22}][T_{<k>}(L-1)]^T$).

Gemäß (6.22) und (6.23) lassen sich die P_{ij} (i,j = 1,2) wie folgt herleiten, wobei $X_{f<k>}$ zur Vereinfachung durch X_b ersetzt wird:

$$\begin{aligned}&[P_{11}] = [T_{<k>}(L-1)][U_{11}][T_{<k>}(L-1)]^T \\ &= [T_{<k>}(L-1)]\{[X_{WH11}(L-1)]+[X_{WH12}(L-1)]+[X_{WH21}(L-1)]+[X_{WH22}(L-1)]\}[T_{<k>}(L-1)]^T \\ &= [X_{b11}(L-1)] + [X_{b12}(L-1)] + [X_{b21}(L-1)] + [X_{b22}(L-1)],\end{aligned}$$

$$[P_{12}] = [T_{<k>}(L-1)][U_{12}][T_{<k>}(L-1)]^T$$
$$= [T_{<k>}(L-1)]\{[X_{WH11}(L-1)]-[X_{WH12}(L-1)]+[X_{WH21}(L-1)]-[X_{WH22}(L-1)]\}[T_{<k>}(L-1)]^T$$
$$= [X_{b11}(L-1)] - [X_{b12}(L-1)] + [X_{b21}(L-1)] - [X_{b22}(L-1)],$$

$$[P_{21}] = [T_{<k>}(L-1)][U_{21}][T_{<k>}(L-1)]^T$$
$$= [T_{<k>}(L-1)]\{[X_{WH11}(L-1)]+[X_{WH12}(L-1)]-[X_{WH21}(L-1)]-[X_{WH22}(L-1)]\}[T_{<k>}(L-1)]^T$$
$$= [X_{b11}(L-1)] + [X_{b12}(L-1)] - [X_{b21}(L-1)] - [X_{b22}(L-1)],$$

$$[P_{22}] = [T_{<k>}(L-1)][U_{22}][T_{<k>}(L-1)]^T$$
$$= [T_{<k>}(L-1)]\{[X_{WH11}(L-1)]-[X_{WH12}(L-1)]-[X_{WH21}(L-1)]+[X_{WH22}(L-1)]\}[T_{<k>}(L-1)]^T$$
$$= [X_{b11}(L-1)] - [X_{b12}(L-1)] - [X_{b21}(L-1)] + [X_{b22}(L-1)].$$

Damit erhalten wir

$$[X_{f<k>}(L)] = \begin{bmatrix} [P_{11}] & [P_{12}][Z_{<k>}(L-1)]^T \\ [Z_{<k>}(L-1)][P_{21}] & [Z_{<k>}(L-1)][P_{22}][Z_{<k>}(L-1)]^T \end{bmatrix}.$$

So ergibt sich schließlich

$$[X_{f<k>}(L)] = \begin{bmatrix} [P_{11}] & [P_{12}][P]^T \\ [P][P_{21}] & [P][P_{22}][P^T] \end{bmatrix},$$

wobei $[P_{11}] = [X_{b11}(L-1)] + [X_{b12}(L-1)] + [X_{b21}(L-1)] + [X_{b22}(L-1)],$
$[P_{12}] = [X_{b11}(L-1)] - [X_{b12}(L-1)] + [X_{b21}(L-1)] - [X_{b22}(L-1)],$
$[P_{21}] = [X_{b11}(L-1)] + [X_{b12}(L-1)] - [X_{b21}(L-1)] - [X_{b22}(L-1)],$
$[P_{22}] = [X_{b11}(L-1)] - [X_{b12}(L-1)] - [X_{b21}(L-1)] + [X_{b22}(L-1)],$

$$[P] = [R_{<k>}(L-1)][T_{<k>}(L-1)]^T = [Z_{<k>}(L-1)].$$

Damit ist gezeigt, wie sich die 2D-Fourier-Matrix rekursiv aus ihren Untermatrizen unter Beachtung der zweidimensionalen Bitumordnung erzeugen läßt. Ein streng "hierarchischer" Aufbau ist demnach für die DFT nicht gegeben. Ein Beispiel soll die rekursive Prozedur verdeutlichen.

Im Abschnitt 4.2.6 hatten wir erhalten:

$$[T_k(1)] = \left[\begin{array}{c|c} T_{<k>}(0) & 0 \\ \hline 0 & R_{<k>}(0) \end{array}\right] = \begin{bmatrix} \underline{T}_0(1) \\ \\ \underline{T}_1(1) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \\ 0 & 1 \end{bmatrix},$$

$$[T_k(2)] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

sowie

$$[R_{<k>}(1)] = \begin{bmatrix} (1-j)/2 & (1+j)/2 \\ \\ (1+j)/2 & (1-j)/2 \end{bmatrix}$$

und

$$[T_{<k>}(2)] = \left[\begin{array}{cc|cc} 1 & 0 & \multicolumn{2}{c}{0} \\ 0 & 1 & & \\ \hline & & (1-j)/2 & (1+j)/2 \\ 0 & & (1+j)/2 & (1-j)/2 \end{array}\right] = \left[\begin{array}{c|c} T_{<k>}(1) & 0 \\ \hline 0 & R_{<k>}(1) \end{array}\right].$$

Gemäß Gleichung (6.25) gilt

$$[Z_{<k>}(1)] = [R_{<k>}(1)][T_{<k>}(1)]^T = [Z_{<k>}(1)]^T.$$

Nun betrachten wir ein Beispiel für N=4 (L=2). Aus den Definitionen von P'_{ij} und P_{ij} mit

$$A = (1-j)/2, \quad B = (1+j)/2 \quad \text{und} \quad [X_{WH}(L)] = \begin{bmatrix} [V_{11}] & [V_{12}] \\ \\ [V_{21}] & [V_{22}] \end{bmatrix},$$

und

$$[V_{11}] = \begin{bmatrix} U_0 & U_1 \\ \\ U_4 & U_5 \end{bmatrix}, \quad [V_{12}] = \begin{bmatrix} U_2 & U_3 \\ \\ U_6 & U_7 \end{bmatrix}, \quad [V_{21}] = \begin{bmatrix} U_8 & U_9 \\ \\ U_{12} & U_{13} \end{bmatrix}, \quad [V_{22}] = \begin{bmatrix} U_{10} & U_{11} \\ \\ U_{14} & U_{15} \end{bmatrix}$$

erhält man

$$[P'_{11}] = [P_{11}] = [T_{<k>}(1)][V_{11}][T_{<k>}(1)]^T =$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} U_0 & U_1 \\ U_4 & U_5 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} U_0 & U_1 \\ U_4 & U_5 \end{bmatrix}.$$

$$[P'_{12}] = [T_{<k>}(1)][V_{12}][R_{<k>}(1)]^T$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} U_2 & U_3 \\ U_6 & U_7 \end{bmatrix} \begin{bmatrix} (1-j)/2 & (1+j)/2 \\ (1+j)/2 & (1-j)/2 \end{bmatrix} = \begin{bmatrix} AU_2+BU_3 & BU_2+AU_3 \\ AU_6+BU_7 & BU_6+AU_7 \end{bmatrix}.$$

$$[P'_{21}] = [R_{<k>}(1)][V_{21}][T_{<k>}(L)]^T$$

$$= \begin{bmatrix} (1-j)/2 & (1+j)/2 \\ (1+j)/2 & (1-j)/2 \end{bmatrix} \begin{bmatrix} U_8 & U_9 \\ U_{12} & U_{13} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} AU_8+BU_{12} & AU_9+BU_{13} \\ BU_8+AU_{12} & BU_9+AU_{13} \end{bmatrix}.$$

$$[P'_{22}] = [R_{<k>}(1)][V_{22}][R_{<k>}(1)]^T$$

$$= \begin{bmatrix} (1-j)/2 & (1+j)/2 \\ (1+j)/2 & (1-j)/2 \end{bmatrix} \begin{bmatrix} U_{10} & U_{11} \\ U_{14} & U_{15} \end{bmatrix} \begin{bmatrix} (1-j)/2 & (1+j)/2 \\ (1+j)/2 & (1-j)/2 \end{bmatrix}$$

$$= \begin{bmatrix} A^2U_{10}+ABU_{14}+ABU_{11}+B^2U_{15} & ABU_{10}+B^2U_{14}+A^2U_{11}+BAU_{15} \\ ABU_{10}+A^2U_{14}+B^2U_{11}+ABU_{15} & B^2U_{10}+ABU_{14}+ABU_{11}+A^2U_{15} \end{bmatrix}.$$

Dann folgt aus (6.25)

$$[X_{f<k>}(2)] =$$

$$\left[\begin{array}{c|c|c|c} U_0 & U_1 & AU_2+BU_3 & BU_2+AU_3 \\ U_4 & U_5 & AU_6+BU_7 & BU_6+AU_7 \\ \hline AU_8+BU_{12} & AU_9+BU_{13} & A^2U_{10}+ABU_{14}+ABU_{11}+B^2U_{15} & ABU_{10}+B^2U_{14}+A^2U_{11}+BAU_{15} \\ BU_8+AU_{12} & BU_9+AU_{13} & ABU_{10}+A^2U_{14}+B^2U_{11}+ABU_{15} & B^2U_{10}+ABU_{14}+ABU_{11}+A^2U_{15} \end{array}\right]. \quad (6.27)$$

Wir betrachten nun die direkte Berechnung aus der Beziehung zwischen 2D-DFT und 2D-WHT. Aus der Gleichung (6.22) folgt

$$[X_{f<k>}(2)] = [T_{<k>}(2)][X_{WH}(2)][T_{<k>}(2)]^T$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & B & A \end{bmatrix} \begin{bmatrix} U_0 & U_1 & U_2 & U_3 \\ U_4 & U_5 & U_6 & U_7 \\ U_8 & U_9 & U_{10} & U_{11} \\ U_{12} & U_{13} & U_{14} & U_{15} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & B & A \end{bmatrix}$$

$$= \left[\begin{array}{c|c|c|c} U_0 & U_1 & AU_2+BU_3 & BU_2+AU_3 \\ U_4 & U_5 & AU_6+BU_7 & BU_6+AU_7 \\ \hline AU_8+BU_{12} & AU_9+BU_{13} & A^2U_{10}+ABU_{14}+ABU_{11}+B^2U_{15} & ABU_{10}+B^2U_{14}+A^2U_{11}+BAU_{15} \\ BU_8+AU_{12} & BU_9+AU_{13} & ABU_{10}+A^2U_{14}+B^2U_{11}+ABU_{15} & B^2U_{10}+ABU_{14}+ABU_{11}+A^2U_{15} \end{array}\right].$$

So ist gezeigt, daß das Ergebnis der direkten Berechnung mit dem der Gleichung (6.27) übereinstimmt.

6.2.5 Vereinheitlichte Behandlung der zweidimensionalen DFT-Algorithmen

In diesem Abschnitt besprechen wir eine Verallgemeinerung des Cooley-Tukey-Algorithmus', die das Berechnen von zweidimensionalen entweder im Objektbereich oder im Fourier-Bereich periodisch abgetasteten Signalen ermöglicht [5.2], [6.1]. Dieser verallgemeinerte Algorithmus vereinigt die konventionellen rechteckigen Zeile/Spalte- und die Vektor-Radix-Algorithmen. Man erhält diesen FFT-Algorithmus durch die Faktorisierung einer ganzzahligen Matrix.

Die 2D-Transformationen werden im allgemeinen entweder mit Hilfe der Zeile/Spalte-Zerlegung der DFT-Summen, die die zweidimensionalen DFT-Berechnungen in die Berechnung von zwei eindimensionalen DFT aufteilen, oder mit Hilfe des Vektor-Radix-Algorithmus ausgeführt. Tatsächlich sind die Zeile/Spalte-Zerlegung und Vektor-Radix-Algorithmus nur spezielle Fälle eines all-

gemeinen Algorithmus' der viele andere Spezialfälle enthält. Natürlich unterscheiden sich diese alternativen FFT-Algorithmen in ihren Berechnungskomplexitäten. Es ist bekannt, daß die Leistungsfähigkeit des eindimensionalen FFT-Algorithmus' sehr stark von N abhängt. Diese Algorithmen sind dann sehr effizient, wenn N eine aus vielen Faktoren zusammengesetzte ganze Zahl ist. Das Fundamentalgesetz der Algebra besagt, daß die Primfaktorzerlegung von ganzen Zahlen bis auf die Anordnung der Faktoren eindeutig ist. Ein derartiges Theorem für den zweidimensionalen Fall existiert jedoch nicht.

Das zweidimensionale Gegenstück zu N ist eine ganzzahlige Matrix [N], die Periodizitätsmatrix heißt. Weil die Primfaktorzerlegung einer ganzzahligen Matrix nicht eindeutig ist, kann man mehrere unterschiedliche FFT-Algorithmen herleiten, deren Effektivitäten unterschiedlich sind. Zwei spezielle Faktorzerlegungen führen auf den Zeile/Spalte-Algorithmus und den Vektor-Radix-Algorithmus.

Eine periodische zweidimensionale Sequenz ist eine, die sich in rechteckigen Intervallen in jeder von zwei unabhängigen Richtungen wiederholt. Eine zweidimensionale Sequenz $\tilde{x}(n_1,n_2)$ ist periodisch, wenn sie die Eigenschaft

$$\tilde{x}([n]+[N][e]) = \tilde{x}[n]$$

für alle ganzzahligen Vektoren [n] und [e] hat und die ganzzahlige (2x2)-Matrix [N] eine nicht verschwindende Determinante hat. Dabei betont die Tilde die Periodizität von x([n]). Abweichend von der sonstigen Bezeichnungsweise stellen wir in diesem Unterabschnitt (Spalten-)Vektoren durch Kleinbuchstaben in eckigen Klammern dar. Eine zweidimensionale periodische Sequenz wiederholt sich in den beiden Richtungen derart, daß die Spalten der Periodizitätsmatrix [N] die Periodizität in diesen Richtungen beschreibt. Wenn [N] diagonal ist, bezeichnet man die Sequenz als rechteckig periodisch. Für Periodizitätsmatrizen ist die Determinante det[N] ganzzahlig und ihr Absolutwert ist gleich der Anzahl der Abtastungen in einer Periode von $\tilde{x}([n])$.

Jede periodische Sequenz $\tilde{x}([n])$ mit der Periodizitätsmatrix [N] kann durch die Menge ihrer Fourier-Koeffizienten exakt repräsentiert werden, d.h.

$$\tilde{x}([n]) = |\det[N]|^{-1} \sum_{[k]\in J_{[N]}} \tilde{X}_f([k])\, \exp\{j[k]^T(2\pi/N)[n]\},$$

$$\tilde{X}_f([k]) = \sum_{[n]\in I_{[N]}} \tilde{x}([n])\ \exp\{-j[k]^T(2\pi/N)[n]\}.$$

Die Sequenz der Koeffizienten $\tilde{x}([k])$ ist periodisch mit der Periodizitätsmatrix $[N]^T$. Die Regionen $I_{[N]}$ bzw. $J_{[N]}$ bezeichnen die Menge der Abtastungen in einer Periode von $\tilde{x}([n])$ bzw. $\tilde{X}_f([k])$.

Man kann eine verallgemeinerte DFT dadurch definieren, daß man die eins-zu-eins Korrespondenz zwischen endlichen Sequenzen und den periodischen Sequenzen der Periodizitätsmatrix [N] mit Hilfe der Menge $I_{[N]}$ benutzt, die die Abtastungen einer Periode enthält. Wenn insbesondere x([n]) eine Sequenz endlich vieler Stützstellen in $I_{[N]}$ bezeichnet, dann ist

$$\tilde{x}([n]) = \sum_{[e]} x([n]+[N][e]),$$

wobei [e] die Werte aller ganzzahligen zweidimensionalen Vektoren annimmt und

$$x([n]) = \begin{cases} \tilde{x}([n]) & \text{für } [n]\in I_{[N]} \\ 0 & \text{sonst} \end{cases}$$

ist.

Deshalb ist x([n]) durch $\tilde{x}([n])$ eindeutig gegeben, das seinerseits durch $\tilde{X}_f([k])$ spezifiziert ist. Die DFT kann dann durch die folgende Beziehung angegeben werden:

$$\begin{aligned} X_f([k]) &= \sum_{[n]\in I_{[N]}} x([n])\exp\{-j[k]^T(2\pi[N]^{-1})[n]\}, && [k]\in J_{[N]}. \\ x([n]) &= |\det[N]|^{-1} \sum_{[k]\in J_{[N]}} X_f([k])\exp\{j[k]^T(2\pi[N]^{-1})[n]\}, && [n]\in I_{[N]}. \end{aligned} \qquad (6.28)$$

2D-Algorithmen vom Cooley-Tukey-Typ existieren für die Berechnung von (6.28) immer dann, wenn [N] in ein nicht-triviales Produkt zweier ganzzahliger Matrizen faktorisiert werden kann. Dies ist in Übereinstimmung mit der bekannten Bedingung für eine eindimensionale Transformation, die eine zusammengesetzte ganzzahlige Transformationslänge benötigt. Wie auch im eindimensionalen Fall, ist die Rechenzeitsparnis umso größer, je größer die Anzahl der

Faktoren in [N] ist. Wenn det[N] keine Primzahl ist, dann enthält eine ganzzahlige Matrix [N] nicht-triviale ganzzahlige Matrixfaktoren. Dann kann [N] in ein Produkt von zwei ganzzahligen Matrizen [P] und [Q] zerlegt werden

$$[N] = [P][Q], \tag{6.29}$$

wobei weder [P] noch [Q] Einheitsmatrizen sind. Die Faktorisierung in (6.29) ist geordnet, jedoch nicht eindeutig.

Zwei ganzzahlige Vektoren [m] und [n] heißen kongruent wenn

$$[m] = [n]+[N][e]$$

für ganzzahlige Vektoren [e] gilt. Die Indizes der Abtastungen einer periodischen Sequenz sind somit kongruent zu den äquivalenten Abtastungen anderer Perioden dieser Sequenz. Insbesondere ist jede Abtastung von x([n]) kongruent zu einer Abtastung aus $I_{[N]}$. Die Bezeichnung

$$[m] = ([n])_{[N]}$$

wird zur Beschreibung des Vektors benutzt, der sowohl kongruent zu [n] ist, als auch in $I_{[N]}$ enthalten ist.

Wenn [N] die Gleichung (6.29) erfüllt, kann irgendein Vektor [n] in $I_{[N]}$ eindeutig als

$$[n] = ([P][q]+[p])_{[N]} \tag{6.30}$$

dargestellt werden, wobei $[p]\in I_{[P]}$ und $[q]\in I_{[Q]}$. $I_{[P]}$ und $I_{[Q]}$ sind Mengen, die jeweils |det[P]| und |det[Q]| Vektoren enthalten. Jedes Vektorpaar (einer aus $I_{[P]}$ und einer aus $I_{[Q]}$) definiert entsprechend (6.30) einen eindeutigen Vektor [n] aus $I_{[N]}$. Analog dazu kann

$$[k]^T = ([v]^T+[m]^T[Q])_{[N]^T} \tag{6.31}$$

geschrieben werden. Dabei ist $[m]\in J_{[P]}$ und $[v]\in J_{[Q]}$. $J_{[P]}$ und $J_{[Q]}$ sind zwei Mengen von Vektoren im Frequenzbereich, deren Elemente alle Elemente der

Menge $J_{[N]}$ erzeugen, wenn sie gemäß (6.31) kombiniert werden. $J_{[P]}$ und $J_{[Q]}$ enthalten jeweils $|\det[P]|$ und $|\det[Q]|$ Elemente.

Unter Verwendung von (6.30) und (6.31) kann die DFT-Summe in (6.28) umgeschrieben werden

$$X_t([Q]^T[m]+[v]) =$$

$$\sum_{[p],[q]} x(([P][q]+[p])_{[N]})\exp\{(-j[v]^T+[m]^T[Q])(2\pi[N]^{-1})([P][q]+[p])\}.$$

Durch Entwicklung des Exponenten wird die Summe in zwei Teile zerlegt:

$$C([p],[v]) = \sum_{[q]\in I_{[Q]}} x(([P][q]+[p])_{[N]})\exp\{-j([v]^T+[m]^T[Q])(2\pi[Q]^{-1}[q])\}, \quad (6.32)$$

$$X_t([Q]^T[m]+[v]) =$$

$$\sum_{[p]\in I_{[P]}} C([p],[v])\ \exp\{-j[v]^T(2\pi[N]^{-1})[p]\}\ \exp\{-j[m]^T(2\pi[P]^{-1})[p]\}. \quad (6.33)$$

Diese Beziehungen repräsentieren die erste Stufe der Zerlegung eines Cooley-Tukey-FFT-Algorithmus' mit Zeitdezimierung. Wenn

$$[P][Q] = [Q][P],$$

kann auch ein analoger Algorithmus mit Frequenzdezimierung hergeleitet werden.

Zum Verständnis der Gleichungen (6.32) und (6.33) betrachte man die Gleichungen einzeln. Die Sequenz

$$x(([P][q]+[p])_{[N]})$$

wird als eine Sequenz über die Variable [q] angesehen. Diese Sequenz ist periodisch mit der Periodizitätsmatrix [Q]. Dann repräsentiert die Summe in (6.32) eine zweidimensionale DFT bezüglich der Periodizitätsmatrix [Q]. Die Region der Stützstellen für diese Sequenz ist $I_{[Q]}$, die als eine Periode von

$$x(([P][q]+[p])_{[N]})$$

gewählt werden muß. Für alle Werte des Vektors [p] muß eine [Q]-DFT berechnet werden. Das bedeutet, daß eine Anzahl von |det[p]| solcher Transformationen zu berechnen ist.

Die Summe in (6.33) zeigt, wie die Ergebniselemente dieser Transformation mit der Matrix [Q] zur Erzeugung einer DFT mit der Matrix [N] kombiniert werden müssen. Die Zahlen C([p],[v]) werden mit den Drehfaktoren

$$\exp\{-j[v]^T(2\pi[N]^{-1})[p]\}$$

multipliziert. Diese Produkte werden in einer Serie von Schmetterlingen gemäß den Matrizen [P] kombiniert. Dazu sind |det[Q]| Schmetterlinge erforderlich. Wenn entweder [P] oder [Q] faktorisierbar ist, kann die jeweilige Menge der DFTs weiter zerlegt werden. Die Vektoren

$$[n] = ([P][q])_{[N]} \tag{6.34}$$

bilden Untermengen von $I_{[N]}$, die durch Abtastung der $I_{[N]}$ mit der Matrix [P] erzeugt werden. Für einen festen Wert von [p] bilden die Abtastungen

$$[n] = ([P][q]+[p])_{[N]}, \qquad [q]\in I_{[Q]}$$

eine partiell geordnete Menge (Coset) bezüglich dieser Untermengen. Weil jedes Coset die Mächtigkeit einer Untermenge hat, gibt es

$$|\det[N]|/|\det[Q]| = |\det[P]|$$

Cosets. Die Elemente jedes Cosets sind kongruent zueinander. Die Region $I_{[P]}$ soll derart ausgewählt werden, daß sie aus jeweils einem der Elemente aus jedem Coset besteht, insgesamt also aus |det[P]| Elementen.

Die Regionen $J_{[P]}$ und $J_{[Q]}$ sind ähnlich ausgewählt. Weil die Fourier-Transformation periodisch in [k] ist (mit der Periodizitätsmatrix $[N]^T$) und weil

$$[N]^T = [Q]^T[P]^T \quad \text{sowie} \quad [k] = [Q]^T[m]+[v]$$

ist, spielt die Matrix $[Q]^T$ im Frequenzbereich eine analoge Rolle wie [P] im Ortsbereich. Entsprechendes gilt für $[P]^T$ und [Q].

Zur Erläuterung des verallgemeinerten Algorithmus' betrachte man die Berechnung der (4×4)-DFT mit der Periodizitätsmatrix [N], in der die folgende Faktorisierung

$$[N] = [P][Q]$$

benutzt wird:

$$[N] = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}, \qquad [P] = \begin{bmatrix} 2 & 2 \\ 1 & -1 \end{bmatrix}, \qquad [Q] = \begin{bmatrix} 1 & 2 \\ 1 & -2 \end{bmatrix}.$$

Das Bild 6.7(a) zeigt die durch die Abtastung der Matrix [P] in vier Cosets geteilte Region $I_{[N]}$.

Die Elemete aller Cosets sind durch unterschiedliche Symbole dargestellt. Man beachte, daß die vier Cosets in der periodischen Fortsetzung die gleiche Geometrie haben. Zum Definieren von $I_{[Q]}$ ist eine Menge von vier Vektoren notwendig, die (6.34) erfüllen. Eine mögliche Menge dieser Vektoren ist

$$I_{[Q]} = \{ (0,0)^T, (1,0)^T, (2,0)^T, (3,0)^T \}.$$

Die Menge $I_{[P]}$ muß aus jeweils einem Element der Cosets bestehen. Eine Möglichkeit ist

$$I_{[P]} = I_{[Q]}$$

zu setzen. Das Bild 6.7(b) stellt die Abtastregion für die DFT dar, wobei $J_{[N]^T}$ die durch die Matrix $[Q]^T$ in vier Cosets geteilt ist. Aus diesem Bild kann man ersehen, daß eine mögliche Wahl für die Mengen J

$$J_{[P]} = \{(0,0)^T, (1,0)^T, (2,0)^T, (3,0)^T\},$$

$$J_{[Q]} = \{(0,0)^T, (0,1)^T, (0,2)^T, (0,3)^T\}$$

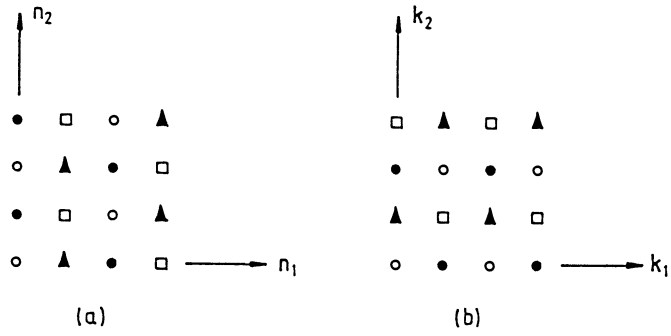


Bild 6.7. a) Stützstellen (Menge $I_{[Q]}$) einer (4×4)-DFT im Ortsbereich, b) Stützstellen (Menge $J_{[P]}$) einer (4×4)-DFT im Frequenzbereich, nach [5.2]

ist. Wenn die vier Mengen gewählt worden sind, kann der Algorithmus bestimmt werden. Jede [Q]-DFT arbeitet auf einem der Eingabe-Cosets. Die Eingabe-Cosets entsprechen dem Bild 6.7(a), um ein Zwischenarray zu erzeugen. Dieses Array wird mit den Drehfaktoren multipliziert, die in diesem Fall alle Eins sind. Die Ergebnisse sind die Eingabemengen der [P]-DFT. Jede dieser DFT erzeugt eines der Ergebnis-Cosets, wie das Bild 6.7(b) zeigt.

Die vier Ergebnisdaten der [Q]-DFT können direkt aus den Eingabedaten berechnet werden. Wenn die Eingabedaten durch x_0, x_1, x_2 und x_3 und die Ausgabedaten durch X_0, X_1, X_2 und X_3 bezeichnet werden, gilt:

$$X_0 = x_0 + x_1 + x_2 + x_3,$$

$$X_1 = x_0 - jx_1 - x_2 + jx_3,$$

$$X_2 = x_0 - x_1 + x_2 - x_3,$$

$$X_3 = x_0 + jx_1 - x_2 - jx_3.$$

Man kann alternativ auch eine andere Faktorisierung von [Q] benutzen:

$$[Q] = \begin{bmatrix} 1 & 2 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}.$$

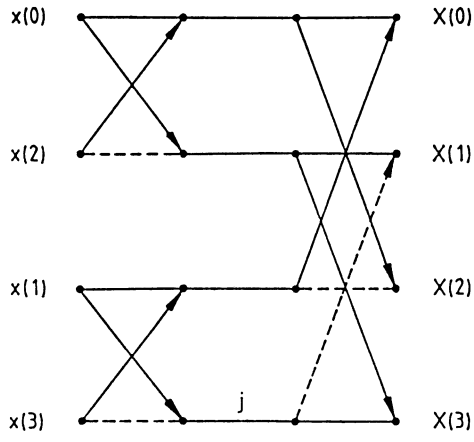


Bild 6.8. SFG für die, [Q]-DFT unter Verwendung der Zerlegung von [Q], nach [5.2]

Der Flußgraph einer 4-Punkte-DFT, der auf diese Faktorisierung basiert, ist im Bild 6.8 dargestellt. Die Eingabedaten und Ergebnisdaten lassen sich in der Weise ordnen, daß der Flußgraph von Bild 6.8 die [Q]-DFT bzw. die [P]-DFT beschreibt.

Die beiden üblichen Algorithmen für die 2D-DFT sind der Zeile/Spalte- und der Vektor-Radix-Algorithmus. Beide können in der Form des verallgemeinerten Cooley-Tukey-Algorithmus' beschrieben werden. Die Periodizitätsmatrix einer zweidimensionalen DFT hat die Form

$$[N] = \begin{bmatrix} N_1 & 0 \\ 0 & N_2 \end{bmatrix}.$$

Mit den Mengen $I_{[N]}$ und $J_{[N]}$ wird die Darstellung der rechteckigen Regionenform der Abtastung gewählt:

$$I_{[N]} = \{(n_1,n_2)^T : 0 \leqq n_1 < N_1,\ 0 \leqq n_2 < N_2\},$$

$$J_{[N]} = \{(k_1,k_2)^T : 0 \leqq k_1 < N_1,\ 0 \leqq k_2 < N_2\}.$$

Der Zeile/Spalte-Algorithmus entspricht der Faktorisierung

$$[N] = [P][Q] = \begin{bmatrix} N_1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & N_2 \end{bmatrix}.$$

Mit dieser Faktorisierung werden die Spaltentransformationen zuerst berechnet, und anschließend die Zeilentransformationen. Wir können die Mengen identifizieren:

$$I_{[P]} = \{(n_1,0)^T,\ 0 \leqq n_1 < N_1\} \qquad I_{[Q]} = \{(0,n_2)^T,\ 0 \leqq n_2 < N_2\}$$

$$J_{[P]} = \{(k_1,0)^T,\ 0 \leqq k_1 < N_1\} \qquad J_{[Q]} = \{(0,k_2)^T,\ 0 \leqq k_2 < N_2\}.$$

Wenn sowohl N_1 wie N_2 durch 2 teilbar sind, hat man eine diagonale Faktorisierung

$$[P][Q] = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} N_1/2 & 0 \\ 0 & N_2/2 \end{bmatrix}.$$

Unter diesen Umständen haben wir

$$I_{[P]} = J_{[P]} = \{(0,0)^T,(0,1)^T,(1,0)^T,(1,1)^T\},$$

$$I_{[Q]} = J_{[Q]} = \{(q_1,q_2),\ 0 \leqq q_1 <(N_1/2),\ 0 \leqq q_2 < (N_2/2)\}.$$

Diese Faktorisierung der Matrix [N] entspricht der ersten Iteration für einen Vektor-Radix-Algorithmus. Die vollständige Zerlegung für

$$N_1 = N_2 = 2^r$$

lautet dann

$$[N] = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \cdots \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

mit r Faktoren.

Der verallgemeinerte Algorithmus besitzt die nützliche Eigenschaft, daß alle die Effizienz des Algorithmus' betreffenden Informationen in der Periodizitätsmatrix enthalten sind. Dies erlaubt den Vergleich und die Optimierung von Algorithmen durch die Überprüfung der Faktorisierung der Matrix [N]. Unter Ausnutzung der möglichen Faktorisierungen kann eine Vielzahl von DFT-Al-

gorithmen (auch für nicht-rechteckige Abtastung) angegeben werden, die jedoch in der Praxis bisher kaum eine Rolle spielen.

6.2.6 DFT-Algorithmen für reelle 2D-Datenarrays

FFT-Algorithmen sind i. allg. für komplexwertige Datenmengen ausgelegt. In der Praxis hat man es jedoch häufig mit reellen Daten zu tun. Im Abschnitt 4.2.7 haben wir die Vereinfachungen dargelegt, die sich für reellen Daten bei der eindimensionalen DFT ergeben. Für den zweidimensionalen Fall lohnt sich ein solches Vorgehen wegen der beträchtlichen Datenmengen noch weitaus mehr. Wir befassen uns deshalb in diesem Abschnitt mit der 2D-DFT reeller Datenarrays, wie sie z.B. bei der digitalen Bildverarbeitung vorkommen.

Für $N_1 = N_2 = N = 2^r$ kann man zunächst die folgenden Beziehungen ausnutzen [6.2]

$$X_f(k_1,k_2) = X_f^*(N-k_1,N-k_2), \qquad k_1,k_2 = 1,2,\ldots,N-1,$$

mit $X_f(k_1,0) = X_f^*(N-k_1,0)$ und $X_f(0,k_2) = X_f^*(0,N-k_2)$.

Bei der Berechnung der 2D-DFT mittels des Zeile/Spalte-Algorithmus' kann man zunächst für die Transformation der (reellen) Zeilendaten einen beliebigen Algorithmus für reelle Daten verwenden. Die so berechneten (Zeilen-) Koeffizienten sind jedoch (außer in den Spalten 0 und N/2) komplexwertig. Deshalb erfordert ihre Verarbeitung nach der Transponierung (außer für die beiden o.e. Spalten) einen Algorithmus, der auch komplexe Daten zuläßt. Alternativ kann man jedoch auch den Algorithmus für reelle Daten auf Real- und Imaginärteil getrennt anwenden, benötigt dann jedoch noch zusätzliche Operationen für die Erzeugung der komplexen 2D-Koeffizienten.

Eine andere Situation liegt vor, wenn die 2D-FFT mit dem Vektor-Radix-Algorithmus berechnet wird, gemäß Abschnitt 6.2.2 bestimmt sich die Radix-2-VR-FFT als

$$X_f(k_1,k_2) = \sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} x(2n_1,2n_2)\, W_{N/2}^{n_1 k_1}\, W_{N/2}^{n_2 k_2} +$$

$$+ W_N^{k1} \sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} x(2n_1+1,2n_2)\, W_{N/2}^{n1k1}\, W_{N/2}^{n2k2}$$

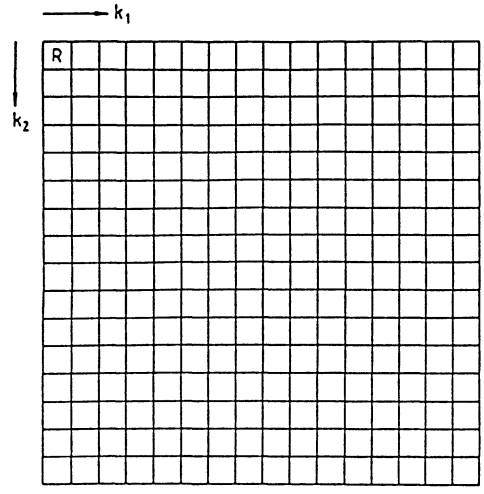
$$+ W_N^{k2} \sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} x(2n_1,2n_2+1)\, W_{N/2}^{n1k1}\, W_{N/2}^{n2k2}$$

$$+ W_N^{k1+k2} \sum_{n_1=0}^{(N/2)-1} \sum_{n_2=0}^{(N/2)-1} x(2n_1+1,2n_2+1)\, W_{N/2}^{n1k1}\, W_{N/2}^{n2k2}.$$

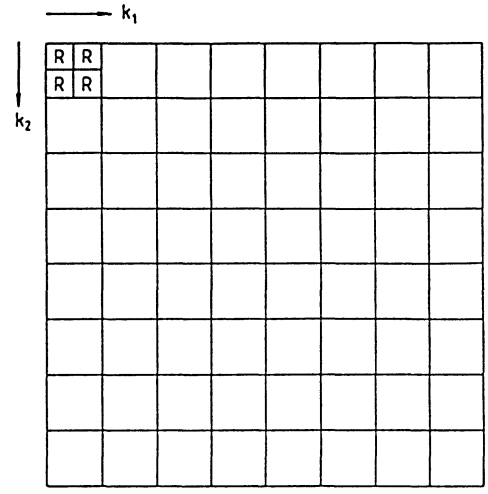
Unabhängig von N ergeben sich jeweils 4 reelle Koeffizienten, nämlich $X_f(0,0)$, $X_f(0,(N/2))$, $X_f((N/2),0)$ und $X_f((N/2),(N/2))$. Alle anderen Koeffizienten sind komplex. Wegen der Beschränkung auf reelle Daten besteht eine 50%-ige Überbestimmung. Deshalb brauchen konjugiert komplexe Paare nur einmal berechnet und abgespeichert werden.

Die Bilder 6.9(a) bis (e) zeigen ein Beispiel für ein (16×16)-Array. Bild 6.9(a) stellt die 16×16 reellen Datenwerte dar. Nach dem i-ten Schritt erhält man Zwischenergebnisse, die als Koeffizienten von jeweils 4^i Daten im Abstand $(N/2)^i$ gedeutet werden können (vgl.Bilder 6.9(b) bis (e)). Außer den o.g. vier reellen Werten ergeben sich jeweils $2^{(N/2)-2(i-1)}$ komplexe Werte, von denen die Hälfte durch ihre konjugiert komplexen Werte bestimmt sind. Von jedem konjugiert komplexen Paar muß also nur ein Wert berechnet und gespeichert werden. Im Bild 6.9 sind diejenigen Koeffizienten $X_f(k_1,k_2)$, die reellwertig sind, mit R bezeichnet. Die übrigen Koeffizienten sind komplex (C und O). Dabei genügt es, die mit C bezeichneten Werte zu speichern, während die mit O ausgewiesenen Koeffizienten sich als konjugiert komplexe Werte der Menge C erweisen und deshalb entbehrlich sind. Die ihnen im "in-place"-Schema eigentlich zukommenden Speicherplätze werden zweckmäßigerweise für die Imaginärteile der Menge C ausgenutzt.

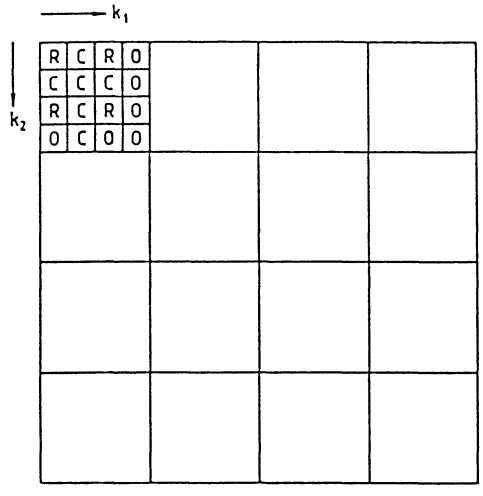
Bild 6.9. Lage der reellen (R), komplexen (C) und redundanten (O) Koeffizienten beim (16×16)-VR-2-Algorithmus: a) Bei Beginn, b) nach der 1. Stufe, c) nach der 2. Stufe, d) nach der 3. Stufe, e) beim Abschluß. Die Muster sind periodisch in (1×1)-, (2×2)-, (4×4)-, ... Arrays


k1
k2
R

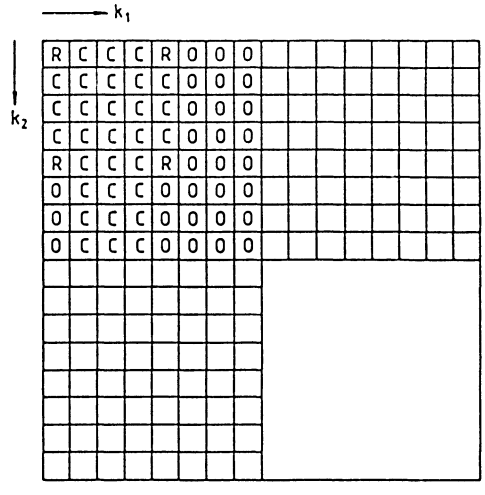
(a)


k1
k2
R R
R R

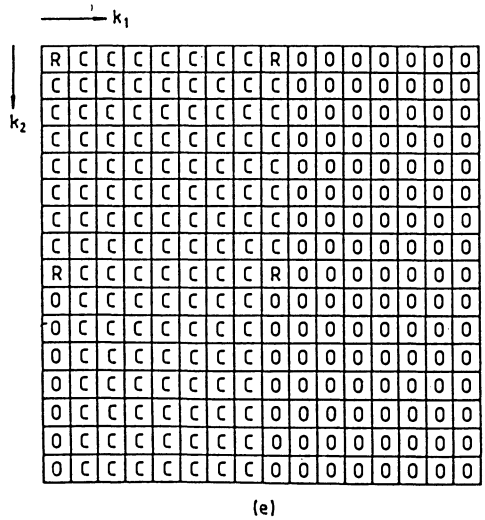
(b)


k1
k2
R C R 0
C C C 0
R C R 0
0 C 0 0

(c)


k1
k2
R C C C R 0 0 0
C C C C C 0 0 0
C C C C C 0 0 0
C C C C C 0 0 0
R C C C R 0 0 0
0 C C C 0 0 0 0
0 C C C 0 0 0 0
0 C C C 0 0 0 0

(d)


k1
k2
R C C C C C C C R 0 0 0 0 0 0 0
C C C C C C C C C 0 0 0 0 0 0 0
C C C C C C C C C 0 0 0 0 0 0 0
C C C C C C C C C 0 0 0 0 0 0 0
C C C C C C C C C 0 0 0 0 0 0 0
C C C C C C C C C 0 0 0 0 0 0 0
C C C C C C C C C 0 0 0 0 0 0 0
C C C C C C C C C 0 0 0 0 0 0 0
R C C C C C C C R 0 0 0 0 0 0 0
0 C C C C C C C C 0 0 0 0 0 0 0
0 C C C C C C C C 0 0 0 0 0 0 0
0 C C C C C C C C 0 0 0 0 0 0 0
0 C C C C C C C C 0 0 0 0 0 0 0
0 C C C C C C C C 0 0 0 0 0 0 0
0 C C C C C C C C 0 0 0 0 0 0 0

(e)

Für die Implementierung benötigt man sieben verschiedene Arten von vereinfachten Schmetterlingen ähnlich Bild 6.6. Tabelle 6.5 gibt an, von welchen Speicherplätzen die Daten zu holen und wo die Ergebnisse abzuspeichern sind.

Tabelle 6.5. Sieben vereinfachte (2×2)-Schmetterlinge (vgl. Bild 6.5) des Vektor-Radix-Algorithmus' für reelle Daten [6.2]

Typ			
1	a	$x(n_1,n_2) + jx((N/2)-n_1,n_2)$	
	A	$X(k_1,k_2) + j(N-k_1,N-k_2)$	
	b	$[x((N/2)+n_1,n_2) + jx(N-n_1,(N/2)-n_2)]\ W^{n1}$	
	B	$X((N/2)+k_1,k_2) + jX((N/2)-k_1,N-k_2)$	
	c	$[x(n_1,(N/2)+n_2) + jx((N/2)-n_1,N-n_2)]\ W^{n2}$	
	C	$[X(N-k_1,(N/2)-k_2) + jX(k_1,(N/2)+k_2)]^*$	
	d	$[x((N/2)+n_1,(N/2)+n_2) + jx(N-n_1,N-n_2)]\ W^{n1+n2}$	
	D	$[X((N/2)-k_1,(N/2)-k_2) + jX((N/2)+k_1,(N/2)+k_2)]^*$	
2	a	$x(n_1,0) + jx((N/2)-n_1,0)$	
	A	$X(k_1,0) + j(N-k_1,0)$	
	b	$[x((N/2)+n_1,0) + jx(N-n_1,0)]\ W^{n1}$	
	B	$[X((N/2)-k_1,0) + jX((N/2)+k_1,0)]^*$	
	c	$x(n_1,(N/2)) + jx((N/2)-n_1,(N/2))$	
	C	$X(k_1,(N/2)) + jX(N-k_1,(N/2))$	
	d	$[x((N/2)+n_1,(N/2)) + jx(N-n_1,(N/2))]\ W^{n1}$	
	D	$[X((N/2)-k_1,(N/2)) + jX((N/2)+k_1,(N/2))]^*$	
3	a	$x(0,n_2) + jx(0,(N/2)-n_2)$	
	A	$X(0,k_2) + j(0,N-k_2)$	
	b	$x((N/2),n_1) + jx((N/2),(N/2)-n_2)$	
	B	$X((N/2),k_1) + jX((N/2),N-k_1$	
	c	$[x(0,(N/2)+n_2) + jx(0,(N/2)-n_2)]\ W^{n2}$	
	C	$[X(0,(N/2)-k_2) + jX(0,(N/2)+k_2)]^*$	
	d	$[x((N/2),(N/2)+n_2) + jx((N/2),N-n_2)]\ W^{n2}$	
	D	$[X((N/2),(N/2)-k_2) + jX((N/2),(N/2)+k_2)]^*$	
4	a	$x(0,0)$	A : $X(0,0)$
	b	$x(N/2,0)$	B : $X(N/2,0)$
	c	$x(0,N/2)$	C : $X(0,N/2)$
	d	$x(N/2,N/2)$	D : $X(N/2,N/2)$
5	a	$x(N/4,N/4)$	A : $X(N/4,N/4)+jX(3N/4,3N/4)$
	b	$jx(3N/4,N/4)$	B : nicht berechnet
	c	$jx(N/4,3N/4)$	C : $[X(3N/4,N/4)+jX(N/4,3N/4)]^*$
	d	$-x(3N/4,3N/4)$	D : nicht berechnet
6	a	$x(N/4,0)$	A : $X(N/4,0)+jX(3N/4,0)$
	b	$jx(3N/4,0)$	B : nicht berechnet
	c	$x(N/4,N/2)$	C : $X(N/4,N/2)+jX(3N/4,N/2)$
	d	$jx(3N/4,N/2)$	D : nicht berechnet
7	a	$x(0,N/4)$	A : $X(0,N/4)+jX(0,3N/4)$
	b	$x(N/2,N/4)$	B : $X(N/2,N/4)+jX(N/2,3N/4)$
	c	$jx(0,3N/4)$	C : nicht berechnet
	d	$jx(N/2,3N/4)$	D : nicht berechnet

Das oben beschriebene Verfahren für reelle 2D-Daten spart beim Vektor-Radix-2-Algorithmus 50% an Speicherplatz und Rechenaufwand gemessen an einem entsprechenden Algorithmus für komplexe Datenarrays. Für die inverse Transformation liegen natürlich komplexe Daten vor. Sie unterliegen jedoch der konjugiert komplexen Symmetrie. Deshalb kann ein Algorithmus entworfen werden, der genau den umgekehrten Weg benutzt.

Bezüglich der Anzahl der Multiplikationen und Additionen wird der VR-2-Algorithmus vom VR-4- und vor allem vom Vektor-Split-Radix-Algorithmus an Effektivität übertroffen [6.2]. Tabelle 6.6 gibt eine Übersicht über die minimal benötigte Anzahl der Operationen.

Tabelle 6.6. Anzahl der Operationen für reelle Vektor-Radix-Algorithmen der Ordnung N×N [6.2]

	VR-2		VR-4		V-S-R	
N	Mult.	Add.	Mult.	Add.	Mult.	Add.
2	0	8			0	8
4	0	52	0	52	0	52
8	24	348			24	348
16	288	2084	216	2012	216	2012
32	2122	11332			1464	10684
64	12672	57732	8640	53700	8280	53340
128	68352	281348			43512	256508
256	345600	1328644	228096	121140	215064	1158108
512	1674240	6130692			1029048	5485500
1024	7870464	27793412	5114880	25037828	4783320	24706268

6.3 Algorithmen für die diskrete 2D-Hartley-Transformation

Die Hartley-Transformation einer zweidimensionalen Funktion $x(n_1,n_2)$ wird als $X_{HY}(k_1,k_2)$ definiert

$$X_{HY}(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1,n_2)\ \mathrm{cas}((2\pi n_1 k_1/N_1)+(2\pi n_2 k_2/N_2)),$$

$$k_1 = 0,1,\ldots,N_1-1, \qquad k_2 = 0,1,\ldots,N_2-1. \tag{6.35}$$

Diese 2D-DHYT und ihre Inverse

$$x(n_1,n_2) = (N_1N_2)^{-1} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X_{HY}(k_1,k_2)\ \mathrm{cas}((2\pi n_1 k_1/N_1)+(2\pi n_2 k_2/N_2)),$$

$$n_1 = 0,1,\ldots,N_1-1, \qquad n_2 = 0,1,\ldots,N_2-1.$$

bilden das Transformationspaar $X_{HY}(k_1,k_2) \longleftrightarrow x(n_1,n_2)$. Die zweidimensionale diskrete Hartley-Transformation ist ein Spezialfall der allgemeinen Transformation (5.4) mit dem Transformationskern

$$g(n_1,n_2,k_1,k_2) = \mathrm{cas}((2\pi n_1 k_1/N_1)+(2\pi n_2 k_2/N_2)).$$

Der Vorteil im Vergleich zu der 2D-DFT liegt darin, daß es zur Berechnung der 2D-DHYT lediglich reeller Arithmetik bedarf.

6.3.1 Zeile/Spalte-Algorithmus der DHYT

Im Gegensatz zur 2D-DFT kann der Transformationskern der 2D-DHYT

$$\mathrm{cas}((2\pi n_1 k_1/N_1)+(2\pi n_2 k_2/N_2))$$

nicht in ein Produkt von zwei Faktoren separiert werden. Wenn der Zeile/Spalte-Algorithmus für die 2D-DHYT gelten soll, muß der Transformationskern durch das Produkt $\mathrm{cas}(2\pi n_1 k_1/N_1)\cdot\mathrm{cas}(2\pi n_2 k_2/N_2)$ ersetzt werden. Eine Methode dazu wäre analog zur 2D-FFT: Zunächst wird die 1D-DHYT der Zeilen und dann der Spalten ausgeführt. Das Ergebnis $X(k_1,k_2)$ wäre dann gegeben durch

$$X(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1,n_2)\ \mathrm{cas}(2\pi n_1 k_1/N_1)\ \mathrm{cas}(2\pi n_2 k_2/N_2),$$

$$k_1 = 0,1,\ldots,N_1-1, \qquad k_2 = 0,1,\ldots,N_2-1.$$

Dies ist jedoch keine Hartley-Transformation. Das Resultat kann jedoch zur gewünschten 2D-DHYT konvertiert werden. Zunächst betrachte man die trigonometrische Identität

$$2\mathrm{cas}(\alpha+\beta) = \mathrm{cas}(\alpha)\mathrm{cas}(\beta)+\mathrm{cas}(\alpha)\mathrm{cas}(-\beta)+\mathrm{cas}(-\alpha)\mathrm{cas}(\beta)-\mathrm{cas}(-\alpha)\mathrm{cas}(-\beta).$$

Die gewünschte 2D-DHYT $X_{HY}(k_1,k_2)$ der Datenmenge $x(n_1,n_2)$ kann als eine Summation von vier Transformationen wie folgt dargestellt werden

$$2X_{HY}(k_1,k_2) = X(k_1,k_2) + X(N_1-k_1,k_2) + X(k_1,N_2-k_2) - X(N_1-k_1,N_2-k_2)$$

$$= X_1 + X_2 + X_3 - X_4.$$

Die 2D-DHYT erhält man dann durch die Kombination von vier Realisierungen von $X(k_1,k_2)$, die auf den Eckpunkten eines Rechtecks liegen. Zur effektiven Berechnung der Kombination ermittelt man gleichzeitig alle vier Terme. Für jede Realisierung wird die Größe

$$D = (1/2)\{(X_1+X_4) - (X_2+X_3)\}$$

ermittelt. Dann werden die Substitutionen

$$X_1 - D \longrightarrow X_1, \qquad X_2 + D \longrightarrow X_2,$$

$$X_3 + D \longrightarrow X_3 \quad \text{und} \quad X_4 - D \longrightarrow X_4$$

ermittelt, um die $X(k_1,k_2)$ in $X_{HY}(k_1,k_2)$ zu konvertieren. Die Größe D ist Null, wenn $k_1=0$, $k_2=0$ oder $k_1= N_1/2$, $k_2= N_2/2$ Folglich braucht die Substitution lediglich von 1 bis $(N_1/2)-1$ und von 1 bis $(N_2/2)-1$ durchgeführt zu werden, um k_1 bzw. k_2 zu erhalten. Diese Diskussion zeigt, daß die 2D-DHYT trotz des nicht separierbaren Transformationskerns durch den Zeile/Spalte-Algorithmus berechenbar ist [6.3].

6.3.2 Vektor-Radix-Algorithmus der DHYT

Die 2D-DHYT einer 2D-Datenmenge ist durch (6.35) definiert. In diesem Abschnitt wird der Vektor-Radix-Algorithmus der DHYT diskutiert, der auf den Abtastwerten von $x(n_1,n_2)$ basiert, die wie folgt indiziert sein können:

n_1 gerade und n_2 gerade,
oder n_1 gerade und n_2 ungerade,
oder n_1 ungerade und n_2 gerade,
oder n_1 ungerade und n_2 ungerade.

Die entsprechenden rekursiven Zerlegungen in vier Summationen bilden den Vektor-Radix-Algorithmus [6.4]. Zur Veranschaulichung schreibt man die Gleichung (6.35) wie folgt um:

$$X_{HY}(k_1,k_2) =$$

$$\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x(n_1,n_2)\{\cos((2\pi n_1k_1/N_1)+(2\pi n_2k_2/N_2))+\sin((2\pi n_1k_1/N_1)+(2\pi n_2k_2/N_2))\},$$

$$k_1 = 0,1,\ldots,N_2-1, \qquad k_2 = 0,1,\ldots,N_1-1. \tag{6.36}$$

Die o.g. Zerlegungsstrategie wird auf alle Summationen angewandt, bis zu deren Grenze von $N_1 = N_2 = N = 2^r$ für die Indizies n_1 und n_2. Damit ergibt sich

$$\begin{aligned}X_{HY}(k_1,k_2) &= X_{00}(k_1,k_2)\\ &+ X_{01}(k_1,k_2)\cos(2\pi k_2/N)\; X_{10}((N/2)-k_1,(N/2)-k_2)\sin(2\pi k_2/N)\\ &+ X_{10}(k_1,k_2)\cos(2\pi k_1/N) + X_{10}((N/2)-k_1,(N/2)-k_2)\sin(2\pi k_1/N)\\ &+ X_{11}(k_1,k_2)\cos(2\pi(k_1+k_2)/N) + X_{11}((N/2)-k_1,(N/2)-k_2)\sin(2\pi(k_1+k_2)/N),\end{aligned}$$

wobei

$$X_{00}(k_1,k_2) =$$

$$\sum_{m_1=0}^{(N/2)-1}\sum_{m_2=0}^{(N/2)-1} x(2m_1,2m_2)\{\cos(2\pi(2m_1k_1+2m_2k_2)/N) + \sin(2\pi(2m_1k_1+2m_2k_2)/N)\},$$

$$X_{01}(k_1,k_2) =$$

$$\sum_{m_1=0}^{(N/2)-1} \sum_{m_2=0}^{(N/2)-1} x(2m_1,2m_2+1)\{\cos(2\pi(2m_1k_1+2m_2k_2)/N) + \sin(2\pi(2m_1k_1+2m_2k_2)/N)\},$$

$$X_{10}(k_1,k_2) =$$

$$\sum_{m_1=0}^{(N/2)-1} \sum_{m_2=0}^{(N/2)-1} x(2m_1+1,2m_2)\{\cos(2\pi(2m_1k_1+2m_2k_2)/N) + \sin(2\pi(2m_1k_1+2m_2k_2)/N)\},$$

$$X_{11}(k_1,k_2) =$$

$$\sum_{m_1=0}^{(N/2)-1} \sum_{m_2=0}^{(N/2)-1} x(2m_1+1,2m_2+1)\{\cos(2\pi(2m_1k_1+2m_2k_2)/N) + \sin(2\pi(2m_1k_1+2m_2k_2)/N)\}.$$

Dabei sind die vier Arrays X_{00}, X_{01}, X_{10} und X_{11} periodisch mit (N/2). Unter Verwendung der trigonometrischen Identitäten

$$\cos(2\pi(k+(N/2))/N) = \cos(2\pi k/N+\pi) = -\cos(2\pi k/N),$$
$$\sin(2\pi(k+(N/2))/N) = \sin(2\pi k/N+\pi) = -\sin(2\pi k/N),$$
$$\cos(2\pi(k_1+k_2+N)/N) = \cos(2\pi(k_1+k_2)/N),$$
$$\sin(2\pi(k_1+k_2+N)/N) = \sin(2\pi(k_1+k_2)/N)$$

erhält man $X_{HY}(k_1+(N/2),k_2)$, $X_{HY}(k_1,k_2+(N/2))$ und $X_{HY}(k_1+(N/2),k_2+(N/2))$ ausgedrückt durch X_{00}, X_{01}, X_{10} und X_{11}:

$$X_{HY}(k_1+(N/2),k_2) = X_{00}(k_1,k_2)$$
$$+ X_{01}(k_1,k_2)\cos(2\pi k_2/N) + X_{01}((N/2)-k_1,(N/2)-k_2)\sin(2\pi k_2/N)$$
$$- X_{10}(k_1,k_2)\cos(2\pi k_1/N) + X_{10}((N/2)-k_1,(N/2)-k_2)\sin(2\pi k_1/N)$$
$$- X_{11}(k_1,k_2)\cos(2\pi(k_1+k_2)/N) + X_{11}((N/2)-k_1,(N/2)-k_2)\sin(2\pi(k_1+k_2)/N),$$

$$X_{HY}(k_1,k_2+(N/2)) = X_{00}(k_1,k_2)$$
$$- X_{01}(k_1,k_2)\cos(2\pi k_2/N) + X_{01}((N/2)-k_1,(N/2)-k_2)\sin(2\pi k_2/N)$$
$$+ X_{10}(k_1,k_2)\cos(2\pi k_1/N) + X_{10}((N/2)-k_1,(N/2)-k_2)\sin(2\pi k_1/N)$$
$$- X_{11}(k_1,k_2)\cos(2\pi(k_1+k_2)/N) + X_{11}((N/2)-k_1,(N/2)-k_2)\sin(2\pi(k_1+k_2)/N),$$

$$X_{HY}(k_1+(N/2),k_2+(N/2)) = X_{00}(k_1,k_2)$$
$$- X_{01}(k_1,k_2)\cos(2\pi k_2/N) + X_{01}((N/2)-k_1,(N/2)-k_2)\sin(2\pi k_2/N) -$$

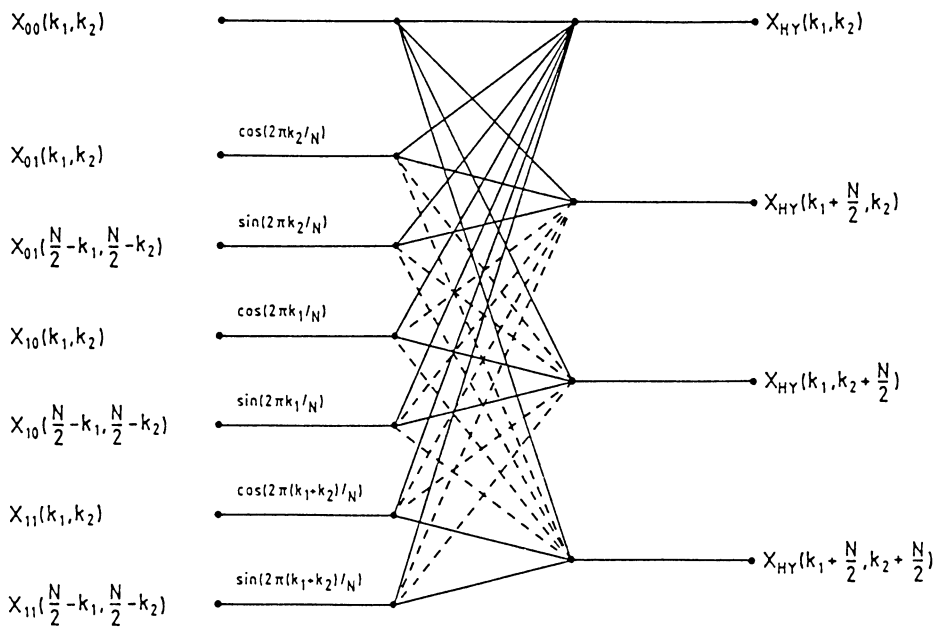


Bild 6.10. Radix-(2×2)-Schmetterling für die Vektor-Radix-2D-DHYT, nach [6.4]

$$- X_{10}(k_1,k_2)\cos(2\pi k_1/N) + X_{10}((N/2)-k_1,(N/2)-k_2)\sin(2\pi k_1/N)$$
$$- X_{11}(k_1,k_2)\cos(2\pi(k_1+k_2)/N) + X_{11}((N/2)-k_1,(N/2)-k_2)\sin(2\pi(k_1+k_2)/N) \quad .$$

Dies illustriert der Signalflußgraph gemäß Bild 6.10. Darin werden 2D-DHYT-Schmetterlinge benutzt, die Radix-4-Strukturen besitzen. Für reelle Daten benötigt jeder Schmetterling 6 reelle Multiplikationen und 8 reelle Additionen. Da der SFG der 2D-DHYT einer (N×N)-Datenmatrix insgesamt $N^2/4$ Stufen enthält, benötigt man zur Berechnung $(3/2)N^2\log_2 N$ reelle Multiplikationen, d.h. lediglich halb so viele wie die Vektor-Radix-DFT, sowie $2N^2\log_2 N$ reelle Additionen. Den SFG dieses Verfahrens zur Berechnung der 2D-DHYT einer reellen (4×4)-Datenmatrix zeigt das Bild 6.11.

Der Vektor-Radix-Algorithmus kann auch vorteilhaft zur Berechnung der zyklischen Faltung von zwei 2D-Datenmengen angewandt werden. Soll die zyklische Faltung $x(n_1,n_2)$ von zwei 2D-Datenmengen $x_1(n_1,n_2)$ und $x_2(n_1,n_2)$ ermittelt werden, so kann man zunächst durch den Vektor-Radix-Algorithmus die 2D-DHYT $X_{HY1}(k_1,k_2)$ und $X_{HY2}(k_1,k_2)$ von $x_1(n_1,n_2)$ bzw. $x_2(n_1,n_2)$ gemäß der Gleichung

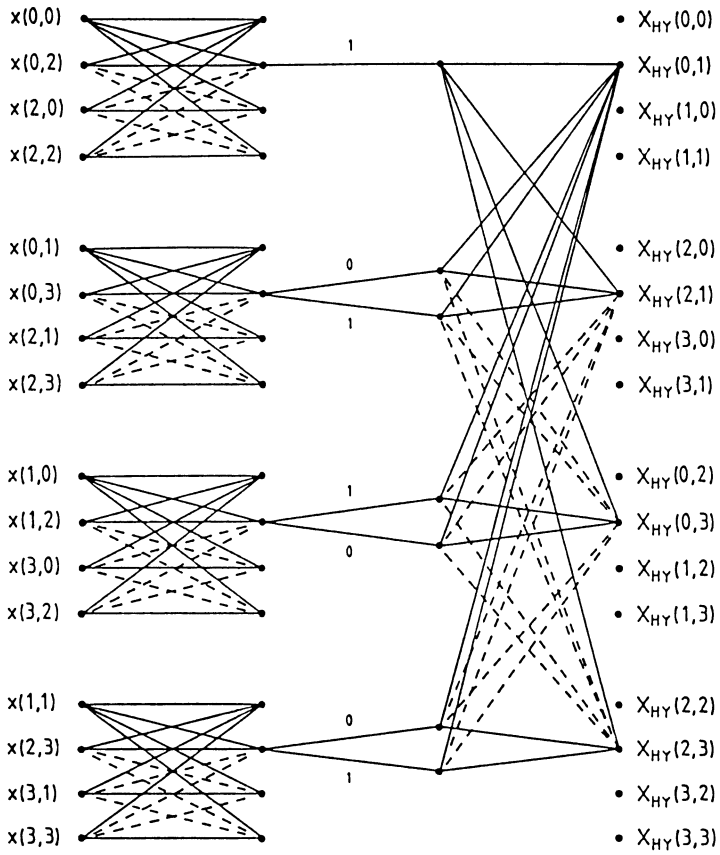


Bild 6.11. (4×4)-Punkte-SFG für eine Radix-(2×2)-DHYT mit nur einem ausführlich gezeichneten Schmetterling, nach [6.4]

(6.36) berechnen. Der Zusammenhang zwischen $X_{HY}(k_1,k_2)$ und $X_{HY1}(k_1,k_2)$ bzw. $X_{HY2}(k_1,k_2)$ ist dann leicht zu ersehen:

$$\begin{aligned} 2X_{HY}(k_1,k_2) &= X_{HY2}(k_1,k_2) + X_{HY2}(N-k_1,N-k_2)X_{HY1}(k_1,k_2) \\ &\quad + X_{HY2}(k_1,k_2) - X_{HY2}(N-k_1,N-k_2)X_{HY1}(N-k_1,N-k_2). \end{aligned}$$

Dann kann man die inverse Hartley-Transformation von $X_{HY}(k_1,k_2)$ berechnen, um die $x(n_1,n_2)$ zu ermitteln.

Abschließend diskutieren wir den Zusammenhang zwischen der 2D-DFT und der 2D-DHYT. Die 2D-DHYT der (N×N)-Datenmenge $x(n_1,n_2)$ kann auch wie folgt geschrieben werden

$$X_{HY}(k_1,k_2) =$$

$$\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\{\cos(2\pi n_1 k_1/N)+\sin(2\pi n_1 k_1/N)\}\{\cos(2\pi n_2 k_2/N)+\sin(2\pi n_2 k_2/N)\}$$

$$k_1,k_2 = 0,1,\ldots,N-1 \quad (6.37)$$

Aus der Definition der 2D-DFT und aus (6.37) erhält man dann

$$2\,\mathrm{Re}\{X_f(k_1,k_2)\} = X_{HY}(N-k_1,k_2) + X_{HY}(k_1,N-k_2) \quad \text{und}$$

$$-2\,\mathrm{Im}\{X_f(k_1,k_2)\} = X_{HY}(k_1,k_2) + X_{HY}(N-k_1,N-k_2).$$

Diese Beziehung kann leicht für den multidimensionalen Fall erweitert werden.

6.4 Algorithmen für die diskrete 2D-Cosinus-Transformation

Die Algorithmen für die zweidimensionale DCT basieren auf der Herleitung über die 2D-DFT, dem Zeile/Spalte-Algorithmus für die DCT, der direkten Berechnung aus der 2D-Datenmenge oder auf der Gewinnung von Teilfolgen unter Verwendung der WHT. Da die beiden ersten Methoden unmittelbar aus den Abschnitten 5.2 und 6.2.1 folgen, befassen wir uns hier mit einem Algorithmus zur direkten Berechnung aus der 2D-Datenmenge und einem Verfahren zur Ermittlung der 2D-DCT-Koeffizienten aus Koeffizienten niedrigerer Ordnung.

Der Zusammenhang zwischen der 2D-DCT und der 2D-DFT einer $(N_1 \times N_2)$-Datenmenge ist gegeben durch

$$X_f(k_1,k_2) = W_{2N1}^{-k1/2}\, W_{2N2}^{-k2/2}\, X_c(k_1,k_2),$$

wobei

$$X_c(k_1,k_2) =$$

$$c(k_1)c(k_2)\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x(n_1,n_2)\cos(\pi(2n_1+1)k_1/(2N_1))\cos(\pi(2n_2+1)k_2/(2N_2)) \quad (6.38)$$

und

$$c(k_1),\ c(k_2) = \begin{cases} 1/\sqrt{2} & \text{wenn } k_1,\ k_2 = 0 \\ 1 & \text{sonst.} \end{cases}$$

Die Basisfunktionen $g(n_1,n_2,k_1,k_2)$ der Transformation sind die abgetasteten Cosinus-Funktionen. Aufgrund des separierbaren Transformationskerns läßt sich (6.38) in Matrixform umschreiben

$$[X_c] = [U^{N1}_{k1,n1}][x][V^{N2}_{k2,n2}],$$

wobei $[U^{N1}_{k1,n1}] = [\cos(\pi(2n_1+1)k_1/(2N_1))]$,

$$[V^{N2}_{k2,n2}] = [\cos(\pi(2n_2+1)k_2/(2N_2))]$$

Die inverse 2D-Cosinus-Transformation

$x(n_1,n_2) =$

$$(N_1N_2)^{-1} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} c(k_1)c(k_2)X_c(k_1,k_2)\cos(\pi(2n_1+1)k_1/(2N_1))\cos(\pi(2n_2+1)k_2/(2N_2))$$

kann entsprechend als

$$[x] = [A^{N1}_{n1,k1}][X_{cv}][B^{N2}_{n2,k2}]$$

mit $[A^{N1}_{n1,k1}] = [\cos(\pi(2n_1+1)k_1/(2N_1))]$,

$$[B^{N2}_{n2,k2}] = [\cos(\pi(2n_2+1)k_2/(2N_2))] \qquad (6.39)$$

und $[X_{cv}(k_1,k_2)] = [c(k_1)c(k_2)\ X_c(k_1,k_2)]$ (6.40)

geschrieben werden.

6.4.1 DCT-Algorithmus zur direkten Anwendung auf die 2D-Datenmenge

In diesem Abschnitt wird ein 2D-FDCT-Algorithmus diskutiert, der nur reelle Multiplikationen und Additionen benötigt. Die Anzahl der reellen Multiplikationen (ausgenommen die Anfangsnormalisierung) ist für $(N_1 \times N_2)$-Datenabtastungen $(3/8)N_1N_2\log_2(N_1 \cdot N_2)$, wobei N_1 und N_2 Potenzen von 2 sind. Verglichen mit der Berechnung der 2D-DCT durch den 1D-FDCT-Algorithmus oder einen komplexen FFT-Algorithmus ist dieser Algorithmus bezüglich der Anzahl der Multiplikationen günstiger. Dazu benutzt dieser 2D-FDCT-Algorithmus die zweidimensionale Datenmenge direkt und nicht getrennt nach Spalten und Zeilen, d.h. er folgt einer ähnlichen Philosophie wie der Vektor-Radix-Algorithmus [6.5].

Die zweidimensionale inverse DCT (IDCT) für Daten des Frequenzbereichs mit den Indizierungen $k_1 = 0,1,\ldots,N_1-1$ und $k_2 = 0,1,\ldots,N_2-1$ läßt sich darstellen durch

$$x(n_1,n_2) = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X_{cv}(k_1,k_2) \cos(\pi(2n_1+1)k_1/(2N_1))\cos(\pi(2n_2+1)k_2/(2N_2)). \tag{6.41}$$

Dabei ist $x(n_1,n_2)$ ein Datenelement im zweidimensionalen Objektbereich, das in Zeile n_1 und Spalte n_2 liegt und $X_c(k_1,k_2)$ ist ein Koeffizient im zweidimensionalen Frequenzbereich, angeordnet in Zeile k_1 und Spalte k_2. Die normalisierte Form der DCT-Koeffizienten ist in (6.40) angegeben.

In Matrixform umgeschrieben wird aus (6.41)

$$[x(n_1,n_2)] = [A^{N_1}(n_1,k_1)][X_{cv}(k_1,k_2)][B^{N2}(n_2,k_2)]. \tag{6.42}$$

wobei $[x]$ und $[X_{cv}]$ die Ordnung $N_1 \times N_2$ besitzen, und die inversen Transformationsmatrizen $[A^{N1}(n_1,k_1)]$ und $[B^{N2}(n_2,k_2)]$ die Ordnung $N_1 \times N_1$ bzw. $N_2 \times N_2$ haben und gemäß (6.39) definiert sind.

Die zweidimensionale DCT gemäß (6.38) kann mit

$$X_{c1}(k_1,k_2) = X_c(k_1,k_2)/(c(k_1)c(k_2))$$

in Summenform folgendermaßen geschrieben werden

$$X_{ci}(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1,n_2)\cos(\pi(2n_1+1)k_1/(2N_1))\cos(\pi(2n_2+1)k_2/(2N_2)),$$

oder in Matrixschreibweise

$$[X_{ci}] = [U^{N1}_{k1,n1}][x][V^{N2}_{k2,n2}]^*, \tag{6.43}$$

wobei $[U^{N1}_{k1,n1}]$ und $[V^{N2}_{k2,n2}]$ die Transponierten von $[A^{N1}_{n1,k1}]$ bzw. $[B^{N2}_{n2,k2}]$ sind, nämlich

$$\begin{aligned}[U^{N1}(k_1,n_1)] &= [\cos(\pi k_1(2n_1+1)/(2N_1))],\\ [V^{N2}(k_2,n_2)] &= [\cos(\pi k_2(2n_2+1)/(2N_2))].\end{aligned} \tag{6.44}$$

Zur Vereinfachung lassen wir die Normalisierungsfaktoren außer Acht. Aus (6.43) und (6.44) ist offensichtlich, daß die zweidimensionale DCT-Matrix die Transponierte der 2D-IDCT-Matrix ist. Wenn der Flußgraph für den 2D-IFDCT-Algorithmus in der umkehrten Richtung durchlaufen wird, ergibt sich deshalb der Flußgraph für den 2D-FDCT-Algorithmus.

Zuerst wird die Datenmatrix des Objektsbereichs behandelt. Die $(N_1 \times N_2)$-Datenmatrix $[x]$ läßt sich aufteilen und in eine Menge von vier $(N_1/2)\times(N_2/2)$-Untermatrizen $[q_1]$, $[q_2]$, $[q_3]$ und $[q_4]$ umordnen

$$[x] = \begin{bmatrix} [q_1] & [q_2] \\ [q_3] & [q_4] \end{bmatrix}. \tag{6.45}$$

wobei für $n_1 = 0,1,\ldots,(N_1/2)-1$ und $n_2 = 0,1,\ldots,(N_2/2)-1$ gilt

$$\begin{aligned}[q_1(n_1,n_2)] &= [x(n_1,n_2)],\\ [q_2(n_1,n_2)] &= [x(n_1,N_2-1-n_2)],\\ [q_3(n_1,n_2)] &= [x(N_1-1-n_1,n_2)],\\ [q_4(n_1,n_2)] &= [x(N_1-1-n_1,N_2-1-n_2)].\end{aligned} \tag{6.46}$$

Die Koeffizientenmatrix im Frequenzbereich ist eine $(N_1 \times N_2)$-Matrix. $[X_{c\,i}(k_1,k_2)]$ läßt sich ebenfalls in vier $(N_1/2)\times(N_2/2)$-Untermatrizen $[Q_1]$, $[Q_2]$, $[Q_3]$ und $[Q_4]$ aufteilen

$$[X_c] = \begin{bmatrix} [Q_1] & [Q_2] \\ [Q_3] & [Q_4] \end{bmatrix}. \tag{6.47}$$

wobei wiederum für $k_1 = 0,1,\ldots,(N_1/2)-1$ und $k_2 = 0,1,\ldots,(N_2/2)-1$ gilt:

$$\begin{aligned} [Q_1(k_1,k_2)] &= [X_c(2k_1,2k_2)], \\ [Q_2(k_1,k_2)] &= [X_c(2k_1,2k_2+1)], \\ [Q_3(k_1,k_2)] &= [X_c(2k_1+1,2k_2)], \\ [Q_4(k_1,k_2)] &= [X_c(2k_1+1,2k_2+1)]. \end{aligned} \tag{6.48}$$

Die $(N_1 \times N_1)$-Transformationsmatrix $[A^{N1}]$ wird geteilt und dann in vier $(N_1/2)\times(N_2/2)$-Untermatrizen $[A_1]$, $[A_2]$, $[A_3]$ und $[A_4]$ gruppiert, wobei die Zeilen dieser Untermatrizen denen der Untermatrizen von $[X_c]$ und die Spalten denen der Untermatrizen von $[X_c]$ entsprechen, d.h.

$$[A^{N1}] = \begin{bmatrix} [A_1] & [A_2] \\ [A_3] & [A_4] \end{bmatrix}$$

$$\begin{aligned} \text{mit}\quad [A_1(n_1,k_1)] &= [A^{N1}(n_1,2k_1)], \\ [A_2(n_1,k_1)] &= [A^{N1}(n_1,2k_1+1)], \\ [A_3(n_1,k_1)] &= [A^{N1}(N_1-1-n_1,2k_1)], \\ [A_4(n_1,k_1)] &= [A^{N1}(N_1-1-n_1,2k_1+1)], \\ n_1 &= 0,1,\ldots,(N_1/2)-1 \text{ und } k_1 = 0,1,\ldots,(N_1/2)-1. \end{aligned} \tag{6.49}$$

Mit (6.39) kann gezeigt werden [6.5], daß $[A_3(n_1,k_1)] = [A_1(n_1,k_1)]$, und $[A_4(n_1,k_1)] = -[A_2(n_1,k_1)]$, d.h.

$$[A^{N1}] = \begin{bmatrix} [A_1] & [A_2] \\ [A_1] & -[A_2] \end{bmatrix}. \tag{6.50}$$

Ähnlich kann die Transformationsmatrix $[B^{N2}]$ in vier Untermatrizen organisiert werden, so daß

$$[B^{N2}] = \begin{bmatrix} [B_1] & [B_2] \\ [B_1] & -[B_2] \end{bmatrix}. \tag{6.51}$$

mit

$$[B_1(n_2,k_2)] = [B^{N1}(n_2,2k_2)] \quad \text{und} \quad [B_2(n_2,k_2)] = [B^{N1}(n_2,2k_2+1)] \tag{6.52}$$

für n_2, $k_2 = 0,1,\ldots,(N_2/2)-1$ und $k_2 = 0,1,\ldots,(N_2/2)-1$. Aus (6.42), (6.45), (6.47), (6.50) und (6.51) erhält man

$$[x] = \begin{bmatrix} [q_1] & [q_2] \\ [q_3] & [q_4] \end{bmatrix} = \begin{bmatrix} [A_1] & [A_2] \\ [A_1] & -[A_2] \end{bmatrix} \begin{bmatrix} [Q_1] & [Q_2] \\ [Q_3] & [Q_4] \end{bmatrix} \begin{bmatrix} [B_1] & [B_2] \\ [B_1] & -[B_2] \end{bmatrix}^T \tag{6.53}$$

Gleichung (6.53) kann nun wie folgt umgeordnet werden:

$$\begin{bmatrix} [q_1] \\ [q_2] \\ [q_3] \\ [q_4] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} [g_1] \\ [g_2] \\ [g_3] \\ [g_4] \end{bmatrix}. \tag{6.54}$$

mit $[g_1] = [A_1][Q_1][B_1]^T$,
$[g_2] = [A_2][Q_2][B_2]^T$,
$[g_3] = [A_3][Q_3][B_3]^T$,
$[g_4] = [A_4][Q_4][B_4]^T$. (6.55)

Gleichung (6.54) zeigt, daß die $(N_1 \times N_2)$-IDCT-Matrix $[x]$ in eine Menge von vier $(N_1/2)\times(N_2/2)$-Untermatrizen $[q_1]$, $[q_2]$, $[q_3]$ und $[q_4]$ aufgeteilt wird, wobei jede eine lineare Kombination (Hadamard-Operation) der vier $(N_1/2)\times(N_2/2)$-Untermatrizen $[g_1]$, $[g_2]$, $[g_3]$ und $[g_4]$ ist.

Es läßt sich nun zeigen, daß jede dieser Untermatrizen eine $(N_1/2)\times(N_2/2)$-IDCT-Matrix ist [6.5]. Aus (6.39) und (6.49) erhält man z.B.

$$[A_1(n_1,k_1)] = [\cos(\pi(2n_1+1)2k_1/(2N_1))] = [A^{N1/2}(n_1,k_1)]$$

oder $[A_1] = [A^{N1/2}]$ für $n_1 = 0,1,\ldots,(N_1/2)-1$ und $k_1 = 0,1,\ldots,(N_1/2)-1$.

Analog dazu erhält man aus (6.39) und (6.52)

$$[B_2(n_2,k_2)] = \cos(\pi(2n_2+1)2k_2/(2N_2)) = [B^{N2/2}(n_2,k_2)]$$

oder $[B_2] = [B^{N2/2}]$ für $n_2 = 0,1,\ldots,(N_2/2)-1$ und $k_2 = 0,1,\ldots,(N_2/2)-1$.

Für die weitere Darstellung werden die diagonalen Matrizen

$$[D_1] = \begin{bmatrix} 1/(2\cos(\pi/(2N_1))) & & \\ & \ddots & \\ & & 1/(2\cos(\pi(N_1-1)/(2N_1))) \end{bmatrix} \tag{6.56}$$

und

$$[D_2] = \begin{bmatrix} 1/(2\cos(\pi/(2N_2))) & & \\ & \ddots & \\ & & 1/(2\cos(\pi(N_2-1)/(2N_2))) \end{bmatrix} \tag{6.57}$$

benötigt.

Zur Vereinfachung der Schreibweise werden folgende Abkürzungen eingeführt:

$$\begin{aligned} [g_1'] &= [g_1] \\ [g_2'] &= [g_2][D_2]^{-1} \\ [g_3'] &= [D_1]^{-1}[g_3] \\ [g_4'] &= [D_1]^{-1}[g_4][D_2]^{-1}. \end{aligned} \tag{6.58}$$

Aus (6.55) folgt

$$[g_1'] = [g_1] = [A^{N1/2}][G_1][B^{N2/2}]^T. \tag{6.59}$$

Gleichung (6.59) beschreibt eine zweidimensonale $(N_1/2)\times(N_2/2)$-IDCT, wobei

$$[G_1] = [Q_1] \text{ oder } [G_1(k_1,k_2)] = [X_c(2k_1,2k_2)]. \tag{6.60}$$

Man beachte, daß die Daten außerhalb der zweidimensionalen ($N_1 \times N_2$)-Blöcke als Null angenommen sind. Analog dazu kann man nun $[G_2]$ und $[G_3]$ herleiten [6.5]:

$$[g_2'] = [g_2][D_2]^{-1} = [A^{N1/2}][G_2][B^{N2/2}]^T, \tag{6.61}$$

$$[g_3'] = [D_1]^{-1}[g_3] = [A^{N1/2}][G_3][B^{N2/2}]^T. \tag{6.62}$$

Das sind zweidimensionale $(N_1/2)\times(N_2/2)$-IDCT für $k_1 = 0,1,\ldots,(N_1/2)-1$ und $k_2 = 0,1,\ldots,(N_2/2)-1$, für die

$$[G_2(k_1,k_2)] = [X_c(2k_1,2k_2+1)] + [X_c(2k_1,2k_2-1)], \tag{6.63}$$

$$[G_3(k_1,k_2)] = [X_c(2k_1+1,2k_2)] + [X_c(2k_1+1,2k_2-1)] \tag{6.64}$$

gilt. $[g_4']$ ist ebenfalls eine zweidimensionale $(N_1/2)\times(N_2/2)$-IDCT-Matrix

$$[g_4'] = [D_1]^{-1}[g_4][D_2]^{-1} = [A^{N1/2}][G_4][B^{N2/2}]^T$$

mit

$$[G_4] = [X_c(2k_1+1,2k_2+1)]+[X_c(2k_1+1,2k_2-1)]+[X_c(2k_1-1,2k_2+1)]+[X_c(2k_1-1,2k_2-1)]. \tag{6.65}$$

Auf diese Weise wird eine zweidimensionale ($N_1 \times N_2$)-IDCT-Matrix in eine Menge von vier $(N_1/2)\times(N_2/2)$-Untermatrizen zerlegt. Jede davon ist eine lineare Kombination von vier, durch zwei diagonale Matrizen $[D_1]$ und $[D_2]$ skalierte $(N_1/2)\times(N_2/2)$-IDCT-Matrizen. Wenn diese Zerlegungsmethode rekursiv durchgeführt wird, ergibt sich ein zweidimensionaler IFDCT-Algorithmus.

Weil die zweidimensionale FDCT die transponierte Form der zweidimensionalen IFDCT ist, sind ihre Flußgraphen gleich denen der zweidimensionalen IFDCT jedoch in umgekehrter Richtung zu durchlaufen. Zur weiteren Erläuterung dieses Algorithmus', betrachte man den Algorithmus in zwei Phasen:

(1) Hintransformation:

Aus den $(N_1/2)\times(N_2/2)$-Untermatrizen $[q_1]$, $[q_2]$, $[q_3]$ und $[q_4]$ nach (6.45) und (6.46) werden die Untermatrizen $[g_1']$, $[g_2']$, $[g_3']$ und $[g_4']$ wie folgt berechnet:

$$\begin{bmatrix} [g_1'] \\ [g_2'] \\ \\ [g_3'] \\ [g_4'] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ & & & \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} [q_1] \\ [q_2] \\ \\ [q_3] \\ [q_4] \end{bmatrix}.$$

Diese Untermatrizen werden durch die diagonalen $(N_1/2)\times(N_2/2)$-Untermatrizen $[D_2]$ und $[D_1]$ gemäß (6.56) und (6.57) in die $(N_1/2)\times(N_2/2)$-Untermatrizen $[g_1]$, $[g_2]$, $[g_3]$ und $[g_4]$ umgeformt:

$$\begin{aligned} [g_1] &= [g_1'], \\ [g_2] &= [g_2'][D_2], \\ [g_3] &= [D_1][g_3'], \\ [g_4] &= [D_1][g_4'][D_2]. \end{aligned}$$

(2) Rücktransformation :

Seien $[G_1']$, $[G_2']$, $[G_3']$ und $[G_4']$ die zweidimensionalen $(N_1/2)\times(N_2/2)$-DCT von $[g_1']$, $[g_2']$, $[g_3']$ und $[g_4']$. Aus den Untermatrizen $[g_1]$, $[g_2]$, $[g_3]$ und $[g_4]$ rekonstruiert man die folgenden $(N_1/2)\times(N_2/2)$-Untermatrizen $[Q_1]$, $[Q_2]$, $[Q_3]$ und $[Q_4]$ gemäß (6.47) und (6.48):

$$\begin{aligned} [Q_1] &= [G_1'], \\ [Q_2] &= [G_2'] + [G_{21}'], \\ [Q_3] &= [G_3'] + [G_{31}'], \\ [Q_4] &= [G_4'] + [G_{41}'] + [G_{42}'] + [G_{43}']. \end{aligned} \tag{6.66}$$

Für $k_1 = 0,1,\ldots,(N_1/2)-1$ und $k_2 = 0,1,\ldots,(N_2/2)-1$ gilt dabei

$$\begin{aligned} [G_{21}'(k_1,k_2)] &= [G_2'(k_1,k_2+1)], \\ [G_{31}'(k_1,k_2)] &= [G_3'(k_1+1,k_2)], \\ [G_{41}'(k_1,k_2)] &= [G_4'(k_1,k_2+1)], \\ [G_{42}'(k_1,k_2)] &= [G_4'(k_1+1,k_2)], \\ \text{und } [G_{43}'(k_1,k_2)] &= [G_4'(k_1+1,k_2+1)]. \end{aligned} \tag{6.67}$$

Es wird angenommen, daß die Daten außerhalb der $(N_1/2)\times(N_2/2)$-Blöcke Null sind.

Als Beispiel betrachten wir eine zweidimensionale (4×4)-IDCT-Matrix.

$$[X_c] = \begin{bmatrix} X_c(0,0) & X_c(0,1) & X_c(0,2) & X_c(0,3) \\ X_c(1,0) & X_c(1,1) & X_c(1,2) & X_c(1,3) \\ X_c(2,0) & X_c(2,1) & X_c(2,2) & X_c(2,3) \\ X_c(3,0) & X_c(3,1) & X_c(3,2) & X_c(3,3) \end{bmatrix} \tag{6.68}$$

wird in vier (2×2)-Untermatrizen $[Q_1]$, $[Q_2]$, $[Q_3]$ und $[Q_4]$ aufgeteilt, die (6.47) und (6.48) erfüllen. Das ergibt

$$[Q_1] = \begin{bmatrix} X_c(0,0) & X_c(0,2) \\ X_c(2,0) & X_c(2,2) \end{bmatrix}, \quad [Q_2] = \begin{bmatrix} X_c(0,1) & X_c(0,3) \\ X_c(2,1) & X_c(2,3) \end{bmatrix},$$

$$[Q_3] = \begin{bmatrix} X_c(1,0) & X_c(1,2) \\ X_c(3,0) & X_c(3,2) \end{bmatrix}, \quad [Q_4] = \begin{bmatrix} X_c(1,1) & X_c(1,3) \\ X_c(3,1) & X_c(3,3) \end{bmatrix}. \tag{6.69}$$

Mit (6.66), (6.67), (6.68) und (6.69) findet man:

$$[G_1] = \begin{bmatrix} X_c(0,0) & X_c(0,2) \\ X_c(2,0) & X_c(2,2) \end{bmatrix},$$

$$[G_2] = \begin{bmatrix} X_c(0,1) & X_c(0,3)+X_c(0,1) \\ X_c(2,1) & X_c(2,3)+X_c(2,1) \end{bmatrix},$$

$$[G_3] = \begin{bmatrix} X_c(1,0) & X_c(1,2) \\ X_c(3,0)+X_c(1,0) & X_c(3,2)+X_c(1,2) \end{bmatrix},$$

$$[G_4] = \begin{bmatrix} X_c(1,1) & X_c(1,3)+X_c(1,1) \\ X_c(3,1)+X_c(1,1) & X_c(3,3)+X_c(3,1)+X_c(1,3)+X_c(1,1) \end{bmatrix}.$$

Man betrachte zuerst nur die Matrix $[G_1]$, die wieder in vier (1×1)–Untermatrizen $[Z_{z1}]$, $[Z_{z2}]$, $[Z_{z3}]$ und $[Z_{z4}]$ aufgespalten werden kann. Mit (6.47), (6.48), (6.60), (6.63), (6.64) und (6.65) erhält man

$$\begin{aligned}
[Z_{z1}(0,0)] &= [G_1(0,0)] = X_c(0,0), \\
[Z_{z2}(0,0)] &= [G_1(0,1)] = X_c(0,2), \\
[Z_{z3}(0,0)] &= [G_1(1,0)] = X_c(2,0), \\
[Z_{z4}(0,0)] &= [G_1(1,1)] = X_c(2,2).
\end{aligned} \tag{6.70}$$

Weil die Transformationsmatrizen $[A_1] = [A(0,0)] = 1$ und $[B_1] = [B(0,0)] = 1$, erhält man aus (6.70) die (1×1)–Matrizen $[z_1']$, $[z_2']$, $[z_3']$ und $[z_4']$. Wenn (6.59), (6.62) und (6.65) benutzt werden, ergibt sich

$$z_1'(0,0) = z_1(0,0) = Z_1(0,0) = X_c(0,0),$$

$$z_2'(0,0) = z_2(0,0)(2\cos(\pi/4))Z_2(0,0) = X_c(0,2)$$
$$\text{oder } z_2(0,0) = X_c(0,2)/(2\cos(\pi/4)),$$

$$z_3'(0,0) = (2\cos(\pi/4))Z_3(0,0) = Z_3(0,0) = X_c(2,0)$$
$$\text{oder } z_3(0,0) = X_c(2,0)/(2\cos(\pi/4)),$$

$$z_4'(0,0) = (2\cos(\pi/4))z_4(0,0)(2\cos(\pi/4)) = Z_4(0,0) = X_c(2,2),$$
$$\text{oder } z_4(0,0) = X_c(2,2)/(4\cos^2(\pi/4)).$$

Dann konstruiert man entsprechend (6.45), (6.47) und (6.54) die (2×2)–Matrix $[g_1']$ aus z_1, z_2, z_3 und z_4:

$$[g_1'] = \begin{bmatrix} g_1'(0,0) \\ g_2'(0,1) \\ \\ g_3'(1,0) \\ g_4'(1,1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \\ z_3 \\ z_4 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} z_1'(0,0) \\ z_2'(0,0)/(2\cos(\pi/4)) \\ \\ z_3'(0,0)/(2\cos(\pi/4)) \\ z_4'(0,0)/(4\cos^2(\pi/4)) \end{bmatrix},$$

d.h.

$$g_1'(0,0) = X_c(0,0) + X_c(0,2)/(2\cos(\pi/4))$$
$$= X_c(2,0)/(2\cos(\pi/4)) + X_c(2,2)/(4\cos^2(\pi/4)),$$

$$g_1'(0,1) = X_c(0,0) - X_c(0,2)/(2\cos(\pi/4))$$
$$= X_c(2,0)/(2\cos(\pi/4)) - X_c(2,2)/(4\cos^2(\pi/4)),$$

$$g_1'(1,0) = X_c(0,0) + X_c(0,2)/(2\cos(\pi/4))$$
$$= X_c(2,0)/(2\cos(\pi/4)) - X_c(2,2)/(4\cos^2(\pi/4)),$$

$$g_1'(1,1) = X_c(0,0) - X_c(0,2)/(2\cos(\pi/4))$$
$$= X_c(2,0)/(2\cos(\pi/4)) \; X_c(2,2)/(4\cos^2(\pi/4)).$$

In analoger Weise kann man die anderen drei (2x2)-Untermatrizen $[g_2']$, $[g_3']$ und $[g_4']$ aus den Untermatrizen $[G_2]$, $[G_3]$ und $[G_4]$ ermitteln. Mit (6.59), (6.58), (6.61) und (6.62) folgt dann

$$[g_1] = \begin{bmatrix} g_1'(0,0) & g_1'(1,0) \\ g_1'(1,0) & g_1'(1,1) \end{bmatrix}.$$

$$[g_2] = [g_2'][D_2] = \begin{bmatrix} g_2'(0,0) & g_2'(1,0) \\ g_2'(1,0) & g_2'(1,1) \end{bmatrix} \begin{bmatrix} 1/(2\cos(\pi/8)) & 0 \\ 0 & 1/(2\cos(\pi/8)) \end{bmatrix}.$$

$$[g_3] = [D_1][g_3'] = \begin{bmatrix} 1/(2\cos(\pi/8)) & 0 \\ 0 & 1/(2\cos(\pi/8)) \end{bmatrix} \begin{bmatrix} g_3'(0,0) & g_3'(1,0) \\ g_3'(1,0) & g_3'(1,1) \end{bmatrix}.$$

$$[g_4] = [D_1][g_4'][D_2]$$

$$= \begin{bmatrix} 1/(2\cos(\pi/8)) & 0 \\ 0 & 1/(2\cos(\pi/8)) \end{bmatrix} \begin{bmatrix} g_4'(0,0) & g_4'(1,0) \\ g_4'(1,0) & g_4'(1,1) \end{bmatrix} \begin{bmatrix} 1/(2\cos(\pi/8)) & 0 \\ 0 & 1/(2\cos(\pi/8)) \end{bmatrix}. \tag{6.71}$$

Mit (6.71) kann man nun die (4x4)-IDCT-Datenmatrix [x] ermitteln. Der entsprechende SFG ist im Bild 6.12 dargestellt. Dazu wendet man (6.45), (6.46) und (6.54) an.

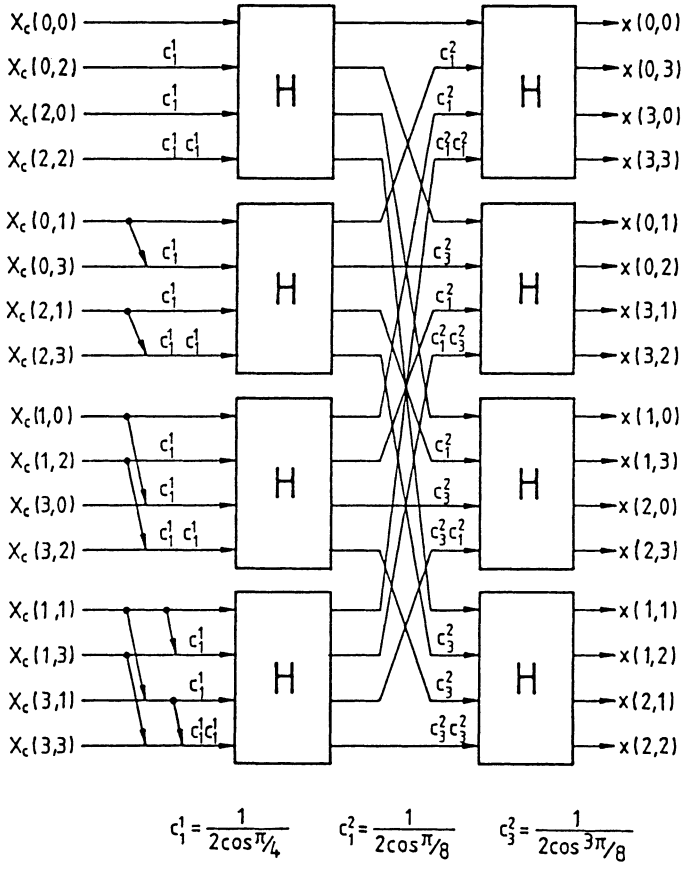


Bild 6.12. SFG einer 2D-IDCT für N×N = 4×4, nach [6.5]

6.4.2 DCT-Berechnung aus Koeffizienten niedrigerer Ordnung

Um den Algorithmus zur Bestimmung der zweidimensionalen DCT-Koeffizienten von Datenblöcken der Größe N×N aus den DCT-Koeffizienten von Datenblöcken (N/2)×(N/2) zu entwickeln, beachte man, daß die zweidimensionale Konversion der Matrix [X] eine Matrix $[T][X][T]^T$ ist, wobei [T] eine orthogonale Transformationsmatrix ist [3.1].

Mit diesen Bezeichnungen kann die Beziehung zwischen der zweidimensionalen Cosinus- und der 2D-Walsh-Hadamard-Transformation (im Bitumkehrform) als

$$[X_c\langle L\rangle] = [T(L)][X_{WH}\langle L\rangle][T(L)]^T \quad L = 0,1,\ldots,r \quad \text{und} \quad N=2^r \tag{6.72}$$

geschrieben werden, wobei $[X_c\langle L\rangle]$ und $[X_{WH}\langle L\rangle]$ die zweidimensionalen DCT- bzw. DWHT-Koeffizienten-Matrizen im Bitumkehrform zur (N×N)-Objektmatrix $[x(n_1,n_2)]$ sind.

Außerdem gelten die Beziehungen

$$[X_{WH}(L)] = [H_H(L)][X_c(L)][H_H(L)]^T \quad \text{und} \quad [X_c(L)] = [C(L)][x(L)][C(L)]^T.$$

Nach der Substitution (6.5) in (6.72) und unter Verwendung der blockdiagonalen Matrix $[T(L)]$ (vgl. die Abschnitte 6.1 und 4.4.6)

$$[T(L)] = \begin{bmatrix} [T(L-1)] & 0 \\ \hline 0 & [R(L-1)] \end{bmatrix}$$

erhält man

$$[X_c(L)] = \begin{bmatrix} [T(L-1)][U_{11}][T(L-1)]^T & [T(L-1)][U_{12}][R(L-1)]^T \\ \hline [R(L-1)][U_{21}][T(L-1)]^T & [R(L-1)][U_{22}][R(L-1)]^T \end{bmatrix}. \qquad (6.73)$$

wobei
$$[U_{11}] = [X_{WH11}(L-1)] + [X_{WH12}(L-1)] + [X_{WH21}(L-1)] + [X_{WH22}(L-1)],$$
$$[U_{12}] = [X_{WH11}(L-1)] - [X_{WH12}(L-1)] + [X_{WH21}(L-1)] - [X_{WH22}(L-1)],$$
$$[U_{21}] = [X_{WH11}(L-1)] + [X_{WH12}(L-1)] - [X_{WH21}(L-1)] - [X_{WH22}(L-1)],$$
$$[U_{22}] = [X_{WH11}(L-1)] - [X_{WH12}(L-1)] - [X_{WH21}(L-1)] + [X_{WH22}(L-1)].$$

Man beachte, daß sich die zweidimensionale DCT des quadratischen Datenblocks (ij) auf die zweidimensionale Walsh-Transformierte desselben Quadranten bezieht, d.h.

$$[X_{cij}\langle L-1\rangle] = [T(L-1)][X_{WHij}\langle L-1\rangle][T(L-1)]^T, \qquad i,j = 1,2. \qquad (6.74)$$

Mit (6.74) und (6.72) schreibt man (6.73) wie folgt um

$$[X_c(L)] = \begin{bmatrix} P'_{11} & P'_{12} \\ P'_{21} & P'_{22} \end{bmatrix},$$

wobei die P'_{ij} die folgende Bedeutung haben :

$$[P'_{11}] = [P_{11}] = [T(L-1)][U_{11}][T(L-1)]^T.$$

$$\begin{aligned}[P'_{12}] &= [T(L-1)][U_{12}][R(L-1)]^T \\ &= [T(L-1)][U_{12}][T(L-1)]^T[T(L-1)][R(L-1)]^T \\ &= [P_{12}][T(L-1)][R(L-1)]^T = [P_{12}][Z(L-1)]^T\end{aligned}$$

(Bemerkung : $[P_{12}] = [T(L-1)][U_{12}][T(L-1)]^T$).

$$\begin{aligned}[P'_{21}] &= [R(L-1)][U_{21}][T(L-1)]^T \\ &= [R(L-1)][T(L-1)]^T[T(L-1)][U_{21}][T(L-1)]^T = [Z(L-1)][P_{21}]\end{aligned}$$

(Bemerkung : $[P_{21}] = [T(L-1)][U_{21}][T(L-1)]^T$).

$$\begin{aligned}[P'_{22}] &= [R(L-1)][U_{22}][R(L-1)]^T \\ &= [R(L-1)][T(L-1)]^T[T(L-1)][U_{22}][T(L-1)]^T[T(L-1)][R(L-1)]^T \\ &= [Z(L-1)][P_{22}][Z(L-1)]^T\end{aligned}$$

(Bemerkung : $[P_{22}] = [T(L-1)][U_{22}][T(L-1)]^T$).

Gemäß (6.72), (6.5) und (6.74) lassen sich nun die P_{ij} wie folgt herleiten:

$$\begin{aligned}[P_{11}] &= [T(L-1)][U_{11}][T(L-1)]^T \\ &= [T(L-1)]\{[X_{WH11}(L-1)]+[X_{WH12}(L-1)]+[X_{WH21}(L-1)]+[X_{WH22}(L-1)]\}[T(L-1)]^T \\ &= [X_{c11}(L-1)] + [X_{c12}(L-1)] + [X_{c21}(L-1)] + [X_{c22}(L-1)].\end{aligned}$$

$$\begin{aligned}[P_{12}] &= [T(L-1)][U_{12}][T(L-1)]^T \\ &= [T(L-1)]\{[X_{WH11}(L-1)]-[X_{WH12}(L-1)]+[X_{WH21}(L-1)]-[X_{WH22}(L-1)]\}[T(L-1)]^T \\ &= [X_{c11}(L-1)] - [X_{c12}(L-1)] + [X_{c21}(L-1)] - [X_{c22}(L-1)].\end{aligned}$$

$$\begin{aligned}[P_{21}] &= [T(L-1)][U_{21}][T(L-1)]^T \\ &= [T(L-1)]\{[X_{WH11}(L-1)]+[X_{WH12}(L-1)]-[X_{WH21}(L-1)]-[X_{WH22}(L-1)]\}[T(L-1)]^T \\ &= [X_{c11}(L-1)] + [X_{c12}(L-1)] - [X_{c21}(L-1)] - [X_{c22}(L-1)].\end{aligned}$$

$$\begin{aligned}[P_{22}] &= [T(L-1)][U_{22}][T(L-1)]^T \\ &= [T(L-1)]\{[X_{WH11}(L-1)]-[X_{WH12}(L-1)]-[X_{WH21}(L-1)]+[X_{WH22}(L-1)]\}[T(L-1)]^T \\ &= [X_{c11}(L-1)] - [X_{c12}(L-1)] - [X_{c21}(L-1)] + [X_{c22}(L-1)].\end{aligned}$$

Damit ergibt sich schließlich

$$[X_c(L)] = \begin{bmatrix} P'_{11} & P'_{12} \\ P'_{21} & P'_{22} \end{bmatrix} = \begin{bmatrix} [P_{11}] & [P_{12}][Z(L-1)]^T \\ [Z(L-1)][P_{21}] & [Z(L-1)][P_{22}][Z(L-1)]^T \end{bmatrix}$$

mit
$$[P_{11}] = [X_{c11}(L-1)] + [X_{c12}(L-1)] + [X_{c21}(L-1)] + [X_{c22}(L-1)],$$
$$[P_{12}] = [X_{c11}(L-1)] - [X_{c12}(L-1)] + [X_{c21}(L-1)] - [X_{c22}(L-1)],$$
$$[P_{21}] = [X_{c11}(L-1)] + [X_{c12}(L-1)] - [X_{c21}(L-1)] - [X_{c22}(L-1)],$$
$$[P_{22}] = [X_{c11}(L-1)] - [X_{c12}(L-1)] - [X_{c21}(L-1)] + [X_{c22}(L-1)]$$

und $[Z(L-1)] = [R(L-1)][T(L-1)]^T$.

Die Anwendung dieses Algorithmus' ist zweckmäßig, wenn die DCT-Koeffizienten von Unterbildern steigender Größe (2x2, 4x4, 8x8,...) gewünscht werden, wie z.B. bei der Transformationskodierung. Bei "in-place"-Verarbeitung ist hierzu normalerweise jeweils eine Rücktransformation zu den Objektdaten nötig. Der hier beschriebene Algorithmus vermeidet dies. Er benötigt $(N^2/4)+(N^3/2)$ Multiplikationen und ist damit bis zu Blockgrößen von 32x32 günstiger als die Methode mit Rücktransformation [3.1].

7 Implementierung der Algorithmen

7.1 Software-Implementierung

Die in den Kapiteln 4 und 6 vorgestellten Algorithmen können in verschiedener Weise realisiert werden. Es ist zu unterscheiden zwischen der softwaremäßigen Implementierung auf verschiedenen Arten von Allzweckrechnern oder auf Signalprozessoren sowie der Realisierung durch eine spezielle Hardware. Die Implementierungen setzen eigentlich erst die Maßstäbe für die Beurteilung der Effizienz eines Algorithmus' in Form von Kosten an Hardware und/oder Berechnungszeit. Eine Bewertung kann deshalb nur unter gleichzeitiger Berücksichtigung der Hardware erfolgen, auf der ein Algorithmus implementiert werden soll.

In zahlreichen Fällen wird eine programmtechnische Implementierung auf einem Allzweckrechner erfolgen, dessen Leistung zwischen der eines einfachen Personal-Computers bis hin zum Vektorrechner oder Supercomputer liegen mag. Für solche Implementierungen liegt die Benutzung einer Hochsprache für die Programmierung natürlich nahe. In der Tat enthalten zahlreiche der im Literaturverzeichnis zitierten Arbeiten derartige Programme. Wegen der meist regelmäßigen Struktur der Signalflußgraphen fallen solche Hochsprachenprogramme i.allg. kurz aus. Allerdings kann man bei Compilern, die nicht für den betreffenden Prozessor optimiert sind, nicht unbedingt davon ausgehen, daß ein optimaler Code erzeugt wird. Deshalb mag es zweckmäßig sein, die Programme teilweise oder ganz in der jeweiligen Assembler-Sprache zu schreiben. Wegen des damit verbundenen Umfangs verzichten wir allerdings bewußt auf die Angabe von Transformationsprogrammen.

Die Programmierung stützt sich auf die Signalflußgraphen und stellt sich im Fall der "in-place"-Verarbeitung besonders einfach dar. Bei der Implementie-

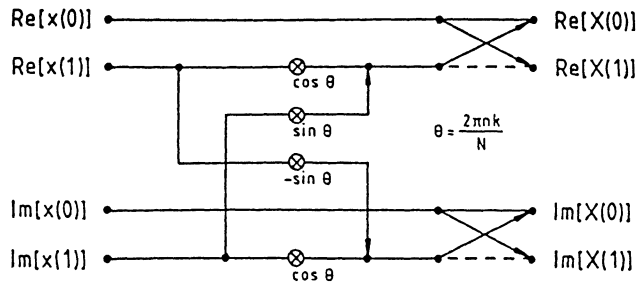


Bild 7.1. Vollständig dargestellter Schmetterling für komplexe Daten und Multiplikation mit komplexen Faktor exp(jθ)

rung der Fourier-Transformation ist zu beachten, daß nicht nur die Koeffizienten sondern i.a. auch die Objektdaten komplex sein können, d.h. in allen Stufen Real- und Imaginärteile verarbeitet werden müssen, die deshalb meist gesondert adressiert werden. Bild 7.1 illustriert die Verarbeitung zweier komplexer Datenwerte X(0) und X(1). Dabei sei mit einem Drehfaktor

$$\exp(-j\theta) = \exp(-j2\pi nk/N)$$

zu multiplizieren. Unter Beachtung von

$$[\mathrm{Re}\{X(1)\}+j\mathrm{Im}\{X(1)\}](\cos\theta-j\sin\theta)$$
$$= [\mathrm{Re}\{X(1\}\cos\theta+\mathrm{Im}\{X(1)\}\sin\theta] + j[\mathrm{Im}\{X(1)\}\cos\theta-\mathrm{Re}\{X(1)\}\sin\theta]$$

erhält man für komplexe Daten und Multiplikatoren einen erweiterten SFG gemäß Bild 7.1. Es zeigt sich, daß eine komplexe Multiplikation 4 reelle Multiplikationen und 2 reelle Additionen erfordert. Außerdem sind noch 4 weitere Additionen bzw. Subtraktionen für die Bildung von Real- und Imaginärteil nötig, sowie 4 Lese- und 4 Schreibzyklen. Diese Operationen sind bei der Abarbeitung *eines* Grundschmetterlings mit Drehfaktor auszuführen.

Die speziellen Eigenschaften eines Rechners bestimmen weitgehend die Effizienz einer Implementierung. So erweisen sich Array-Prozessoren als besonders geeignet. In [7.1] werden Methoden der Implementierung von Radix-2-Transformationen auf Array-Prozessoren betrachtet. Es wird gezeigt, daß eine O(P)-Beschleunigung (P ist die Länge des Arrays) für die arithmetischen Ope-

rationen erhalten wird. Allerdings verursacht das Daten-Routing einen zusätzlichen Organisationsaufwand.

Die Implementierung von Transformationsalgorithmen auf Signalprozessoren ist von Assembler-Programmierung und der Verfügbarkeit von Hardware-Multiplikatoren geprägt [7.2]. Unter Verwendung der (bei vielen Signalprozessoren vorhandenen) Akkumulatoren wird es möglich, Multiplikation und Addition als eine Operation aufzufassen, was zu neuen Optimalitätskriterien und somit zu modifizierten Algorithmen führt [7.3]. Die allgemein übliche Regel, daß die Anzahl der Multiplikationen die Kosten (Rechenzeit) eines Algorithmus' bestimmt, ist hier also außer Kraft gesetzt.

Untersuchungen über die parallele Verarbeitung durch mehrere Prozessoren findet man z.B. in [7.4] bis [7.7]. Allgemein wird angestrebt, die Aufgaben so auf die Prozessoren zu verteilen, daß diese möglichst gleichmäßig ausgelastet werden. Um dieses Ziel zu erreichen, müssen ggf. auch zusätzliche ("in-place"-) Sortierungen vorgenommen werden. Bild 7.2 illustriert diesen Vorgang an einem Beispiel [7.7]. Durch eine "Shuffle"-Sortierung nach der zweiten Iteration wird erreicht, daß vier Prozessoren mit vier privaten Speichern unabhängig voneinander an dem Problem arbeiten können. Da bei fast allen der hier interessierenden Orthogonaltransformationen jeder Koeffizient von allen Objektdaten abhängt, ist eine vollständige Parallelisierung nicht möglich. Es wird also immer Anteile des Algorithmus geben, die sich nicht parallelisieren lassen. In dem im Bild 7.2 dargestellten Beispiel ist das der Abschnitt für die "Perfect-Shuffle"-Umsortierung. Derartig strukturierte Aufgaben, die also seriell abzuarbeitende Anteile enthalten, unterliegen *Amdahls* Gesetz [7.8]. Wenn b der Bruchteil der parallelisierbaren Operationen und p die Anzahl der parallelen Prozesse ist, dann erhält man höchstens eine Beschleunigung um einen Faktor $S = ((1-b) + b/p)^{-1}$. Nimmt man für das obige Beispiel an, daß die Umsortierung 10% des Aufwandes an Operationen erfordert, so erhält man einen Faktor $S = ((1-0,9) + 0,9/4)^{-1} \approx 3,1$ für die Beschleunigung durch vier parallele Prozessoren.

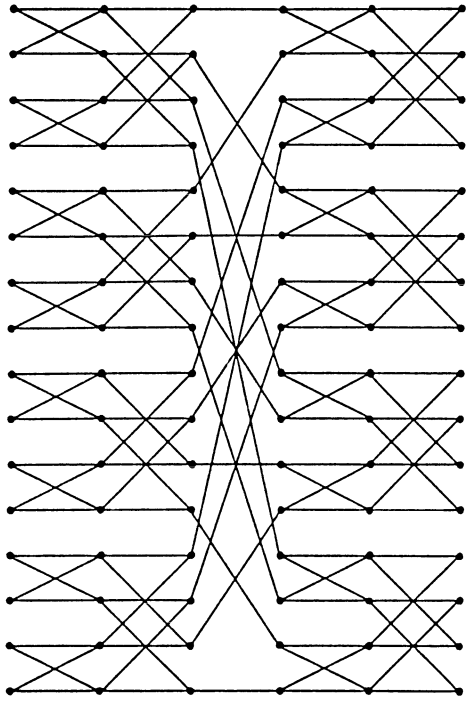


Bild 7.2. Zur Aufteilung einer 16-Punkte-("in-place"-) Transformation auf 4 Prozessoren

7.2 Hardware-Realisierung

Anstatt die Transformationsalgorithmen auf kommerziell verfügbaren Rechnern oder Prozessoren zu implementieren, kann man spezielle Hardware-Strukturen verwenden. Derartige Transformationsprozessoren erweisen sich naturgemaß als äußerst unflexibel (z.B. bezüglich der Transformationslänge). Andererseits erreicht man mit ihnen hohe Ausführungsgeschwindigkeiten bei einem günstigen Preis/Leistungs-Verhältnis. In letzter Zeit sind verschiedene VLSI-Realsierungen bekannt geworden, deren Prinzipien von Pipeline-Architekturen über Parallel-Pipelining bis hin zu zellulären und systolischen Arrays reichen.

Hardware zur schnellen Ausführung von orthogonalen Transformationen kann grob wie folgt klassifiziert werden:

- Serielle Strukturen,
- Pipeline-Strukturen,
- Parallel-Strukturen,
- Array-Strukturen.

Für die serielle Verarbeitung ist es zweckmäßig, die Algorithmen so zu gestalten, daß alle Iterationen von gleicher Struktur sind. Dann kann man sämtliche Iterationen auf der gleichen Hardware ausführen. Bild 7.3 illustriert dies am Beispiel einer 8-Punkte-Transformation. Der SFG zeigt, daß die Operationen nicht "in-place" durchführbar sind und die Reihenfolge von der Bitumkehrordnung abweicht. Multiplikationen (z.B. mit Drehfaktoren) sind nicht dargestellt.

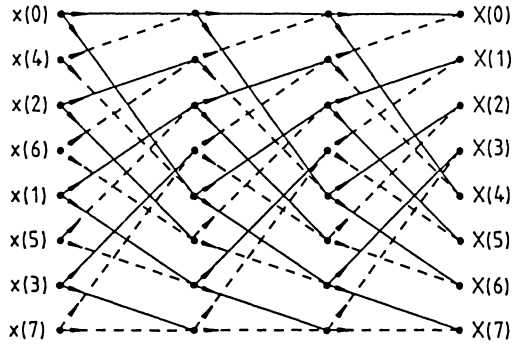


Bild 7.3. SFG mit gleicher Struktur in allen Iterationen

Pipeline-Architekturen beruhen auf der zeitlichen Staffelung der Operationen. Die meist regelmäßige Struktur der Algorithmen von Orthogonaltransformationen erlaubt eine effiziente Anwendung dieses Prinzips [7.9], [7.10]. Eine weitere Steigerung der Geschwindigkeit läßt sich durch eine Kombination von Pipeline- und Parallel-Verarbeitung erzielen. Untersuchungen hierzu findet man in [7.11], [7.12].

Die konsequente Weiterentwicklung des Parallel-/Pipeline-Prinzips führt zur Auflösung der Hardware in ein Array von Prozessoren. Die Verwendung zellulärer Prozessoren erfordert eine Kommutierung der Daten nach jeder Iteration. Für diese Aufgabe eignet sich das "Perfect-Shuffle"-Netzwerk, das im Bild 7.4 an einem Beispiel dargestellt ist. Beispiele für die Anwendung zellulä-

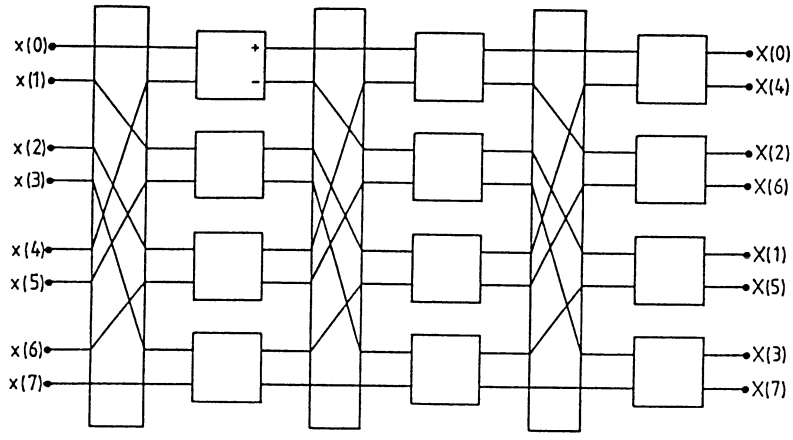


Bild 7.4. Verwendung zellulärer Prozessoren und Kommutatoren

rer Prozessoren auf zweidimensionale Transformationen zeigen die Bilder 7.5 und 7.6. Die Prozessoren erzeugen Summe und Differenz der Inputs. Eine hierarchische WHT, die zunächst die (2x2)-Koeffizienten und aus diesen die (4x4)-Koeffizienten erzeugt, ist im Bild 7.5 dargestellt. Bild 7.6 illustriert eine direkte 2D-FFT. Die Indizierung der Daten folgt dabei Bild 5.8.

Während die zellulären Prozessoren Wörter geeigneter Länge verarbeiten, erfolgt die Verarbeitung bei Systolischen Arrays bitweise. Über die Implementierung der FFT auf Systolischen Arrays berichtet z.B. [7.13].

Vor Einführung der VLSI-Technik war es undenkbar, Hardware-Realisierungen von Orthogonaltransformationen für technisch interessante Werte von N zu vernünftigen Preisen herzustellen. Zum Zeitpunkt der Herausgabe dieses Buches sind entsprechende ICs erhältlich [7.14], [7.15]. Außerdem besteht die Möglichkeit, spezielle Transformationsalgorithmen durch semikundenspezifische ICs (z.B. Gatearrays) zu realisieren. Diese Operationen werden in der Zukunft die Einsatzbereiche orthogonaler Transformationen bedeutend erweitern.

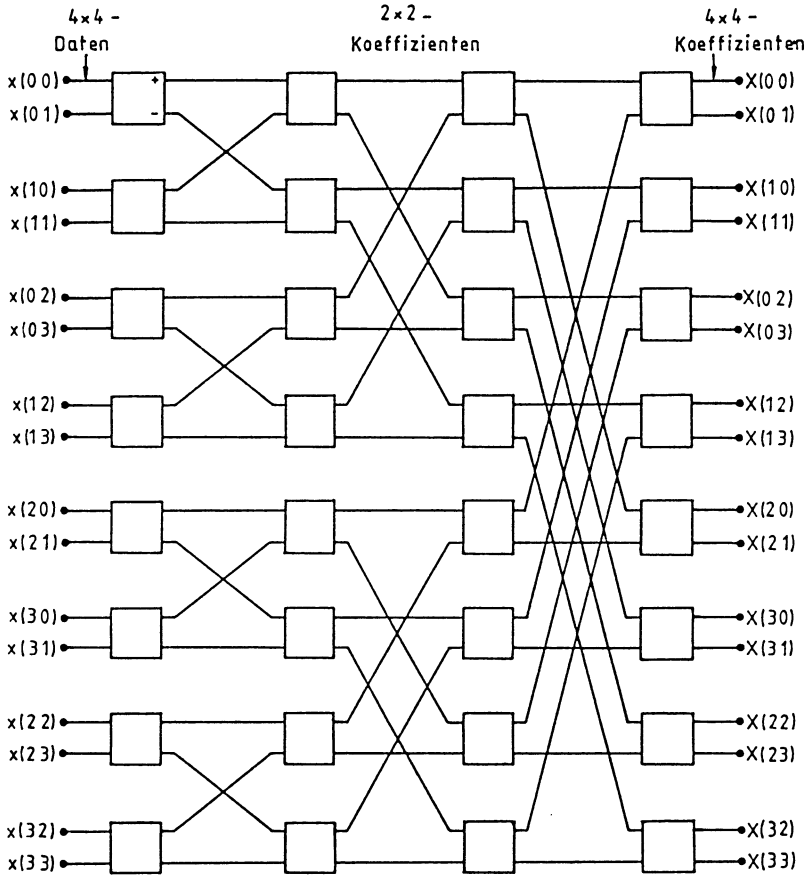


Bild 7.5. Beispiel für die Verwendung zellulärer Prozessoren: hierarchische (4×4)-WHT

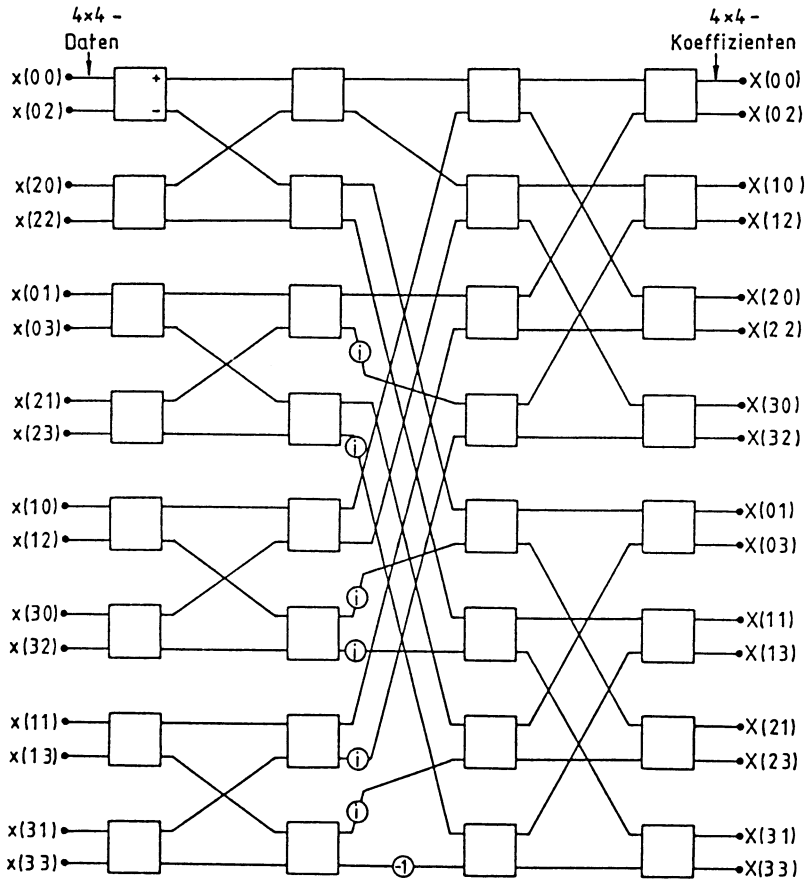


Bild 7.6. Beispiel für die Verwendung zellulärer Prozessoren: direkte (4×4)-FFT

Literaturverzeichnis

Kapitel 1

1.1 Harmuth, H.F. : Transmission of information by orthogonal functions. Berlin, Heidelberg, New York: Springer 1972

1.2 Clarke, R.J. : Transform coding of images. London: Academic Press 1985

1.3 Oppenheim, A.V. ; Schafer, R.W. : Digital signal processing. Englewood Cliffs, N.J. : Prentice-Hall 1975

1.4 Kunt, M. : Digital signal processing. London: Artech House. 1986

1.5 Rabiner, L.R. ; Gold, B. : Theory and application of digital signal processing. Englewood Cliffs, N.J. : Prentice-Hall 1975

1.6 Harmuth, H.F. : Sequency theory : foundations and applications. New York, San Francisco, London: Academic Press 1977

1.7 Gonzales, R.C. ; Wintz, P. : Digital image processing (2.Ed.). Reading, MA: Addison-Wesley 1987

1.8 Rao, K.R. (Ed.): Discrete transforms and their applications. New York: Van Nostrand Reinhold 1985

Kapitel 2

2.1 Brigham, E.O. : FFT - Schnelle Fourier-Transformation. München, Wien: R.Oldenbourg 1985

2.2 Nussbaumer, H.J. : Fast Fourier transform and convolution algorithms. Berlin, Heidelberg, New York: Springer 1981

2.3 Beth, T. : Verfahren der schnellen Fourier-Transformation. B.G. Teubner Stuttgart 1984

2.4 Creutzburg, R. (Ed.): Selected topics on number-theoretic transforms, ZKI-Informationen 2/88. Berlin : Akademie der Wissenschaften der DDR 1988

2.5 Bracewell, R.N. : The fast Hartley transform. Proc. IEEE 72 (1984) 1010-1017

— : The Hartley transform. New York: Oxford University Press 1986

2.6 Elliott, D.F. ; Rao, K.R. : Fast transforms : algorithms, analyses, applications. New York, San Francisco, London: Academic Press 1982

2.7 Ahmed, N. ; Rao, K.R. : Orthogonal transforms for digital signal processing. Berlin, Heidelberg, New York: Springer 1975

2.8 Chen, W. ; Smith, C.H. ; Fralick, S.C. : A fast computational algorithm for the discrete cosine transform. IEEE Trans. Commun. COM-25 (1977) 1004-1009

2.9 Jain, A.K. : A fast Karhunen-Loève transform for a class of stochastic processes. IEEE Trans. Commun. COM-24 (1976) 1023-1029

2.10 Kekre, H.B. ; Solanki, J.K. : Comparative performance of various trigonometric unitary transforms for transform image coding. Int. J. Electronics. 44 (1978) 305-315

2.11 Chrestenson, H.E. : A class of generalized Walsh functions. Pacific J. Math.5 (1955) 17-31

2.12 Liebler, M.E. ; Roesser, R.P. : Multiple-real-valued Walsh functions. In: D.C. Rine (Ed.): Computer science and multiple-valued logic. Amsterdam: North-Holland. (1977) 535-548

2.13 Sato, M. ; Iijima, T. : Pseudo-Hadamard transform for image processing, Architectures and Algorithmus for Digital Image Processing. A.Oosterlinck, P.-E.Danielsson (Eds.) SPIE 27th. Internat. Techn. Symp. (1983) 37-43

2.14 Zhang, G. : Two complete orthogonal sets of real multiple-valued functions. Proc. Internat. Symp. Multiple-Valued Logic (1984) 12-18

2.15 Beauchamp, K.G. : Applications of Walsh and related functions. New York, San Francisco, London: Academic Press 1984

2.16 Niederdrenk, K. : Die endliche Fourier- und Walsh-Transformation mit einer Einführung in die Bildverarbeitung. Braunschweig: Vieweg 1982

2.17 Pratt, W.K. : Digital image processing. New York: J. Wiley & Sons 1978

2.18 Jain, A.K. : Fundamentals of digital image processing. Englewood Cliffs, N.J. : Prentice-Hall 1989

Kapitel 3

3.1 Kashef, B.G. ; Habibi, A. : Direct computation of higher-order DCT coefficents from lower-order DCT coefficients. SPIE Conf. on Applications of Digital Image Processing VII. 504 (1984) 425-431

3.2 Tadokoro, Y. ; Higuchi, T. : Conversion factors from Walsh coefficients to Fourier coefficients. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-31 (1983) 231-232

3.3 Soderstrand, M.A. ; Jenkins, W.K. ; Jullien, G.A. ; Taylor, F.J. (Eds.): Residue number system arithmetic: modern applications in digital signal processing. New York: IEEE Press 1986

3.4 Tseng, B.D. ; Jullien, G.A. ; Miller, W.C. : Implementation of FFT structures using the residue number system. IEEE Trans. Comput. C-28 (1979) 831-844

3.5 Jaroslavskij, L.P. : Einführung in die digitale Bildverarbeitung. Berlin: VEB Deutscher Verlag der Wissenschaften 1985

3.6 Lee, M.H. ; Kaveh, M. : Fast Hadamard transform based on a simple matrix factorization. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-34 (1986) 1666-1667

3.7 Fino, B.J. ; Algazi, V.R. : A unified treatment of discrete fast unitary transforms. SIAM J. Comput. 6 (1977) 700-716

3.8 Burrus,C.S. ; Parks, T.W. : DFT/FFT and convolution algorithms. New York, Chichester, Brisbane, Toronto, Singapore: Wiley 1985

3.9 Blahut, R.E. : Fast algorithms for digital signal processing. Reading, M.A: Addison-Wesley 1987

3.10 Heidemann, M.T. : Multiplicative complexity, convolution, and the DFT. Berlin, Heidelberg, New York: Springer 1988

3.11 Agarwal, R.C. ; Cooley, J.W. : Vectorized mixed radix Fourier transform algorithms. Proc. IEEE 75 (1987) 1283-1292

Kapitel 4

4.1 Cooley, J.W. ; Tukey, J.W. : An algorithm for the machine calculation of complex Fourier series. Math. of Comput. 19 (1965) 297-301

4.2 Cooley, J.W. : The re-discovery of the fast Fourier transform algorithm. Mikrochim. Acta (Wien) III, Springer-Verlag, (1988) 33-45

4.3 Good, I.J. : The interaction algorithm and practical Fourier analysis. J.R.Statist.Soc. B. 20 (1958) 361-372

4.4 Heideman, M.T. ; Johnson, D.H. ; Burrus, C.S. : Gauss and the history of the fast Fourier transformation. IEEE ASSP Magazine. 1 (1984) 14-21

4.5 Stumpff, K. : Grundlagen und Methoden der Periodenforschung. Berlin: Springer 1937
— : Tafeln und Aufgaben zur harmonischen Analyse und Priodogrammrechnung. Berlin: Springer 1939

4.6 Runge, C. ; König, H. : Vorlesungen über numerisches Rechnen (Die Grundlehren der mathematischen Wissenschaften, Band XI). Berlin: Springer 1939

4.7 Duhamel, P. : Implementation of 'split-radix' FFT algorithms for complex, real and real-symmetric data. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-34 (1986) 285-295

4.8 Duhamel, P. ; Hollmann, H. : Split radix FFT algorithm. Electronics Letters. 20 (1984) 14-16

4.9 Sorensen, H.V. ; Heideman, M.T. ; Burrus, C.S. : On computing the split-radix FFT. IEEE Trans. Acoust., Speech und Signal Processing. ASSP-34 (1986) 152-156

4.10 Martens, J.B. : Recursive cyclotomic factorization - a new algorithm for calculating the discrete Fourier transform. IEEE Trans. Acoust., Speech und Signal Processing. ASSP-32 (1984) 750-761

4.11 Vetterli, M. ; Nussbaumer, H.J. : Simple FFT and DCT algorithms with reduced number of operations. Signal Processing. (1984) 267-278

4.12 Wang, Z.D. : Fast algorithms for the discrete W transform and for the discrete Fourier transform. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-32 (1984) 803-816

4.13 Kolba, D.P. ; Parks, T.W. : A prime factor FFT algorithm using high speed convolution. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-25 (1977) 281-294

4.14 Tolimieri, R. ; An, M. ; Lu, C. : Algorithms for discrete Fourier transform and convolution. Berlin, Heidelberg, New York: Springer 1989

4.15 Winograd, S. : On computing the discrete Fourier transform. Math. of Comput. 32 (1978) 175-199

— : Arithmetic complexity of computations. CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM Philadelphia, PA.

4.16 Silverman, H.F. : An introduction to programming the Winograd Fourier transform algorithm (WFTA). IEEE Trans. Acoust., Speech and Signal Processing. ASSP-25 (1977) 152-164

4.17 Tadokoro, Y. ; Higuchi, T. : Diskrete Fourier transform computation via the Walsh transform. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-26 (1978) 236-240

— : Comments on "Diskrete Fourier transform via Walsh transform". IEEE Trans. Acoust., Speech and Signal Processing. ASSP-27 (1979) 295-296

— : Another discrete Fourier transform computation with small multiplications via the Walsh transform. ICASSP'81 Proceedings of the 1981 IEEE International Conference on Acoust., Speech and Signal Processing, 1 (1981) 306-309

4.18 Preuss, R.D. : Very fast computation of the radix-2 discrete Fourier transform. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-30 (1982) 595-607

4.19 Cooley, J.W. ; Lewis, P.A.W. ; Welch, P.D. : The fast Fourier transform: programming considerations in the calculation of 'sine, cosine and Laplace transforms. J. Sound Vib. 12(3) (1970) 315–337

4.20 Storn, R. : Fast algorithms for the discrete Hartley transform. Archiv für Elektronik & Übertragungstechnik. 40 (1986) 233–240

4.21 Pei, S.C. ; Wu, J.L. : Split-radix fast Hartley transform. Electronics Letters. 22 (1986) 26–27

4.22 Hou, H.S. : The fast Hartley transform algorithm. IEEE Trans. Comput. C–36 (1987) 147–156
— : Correction to : The fast Hartley transform algorithm. IEEE Trans. Comput. C–36 (1987) 1135–1136

4.23 Sorensen, H.V. ; Jones, D.L. ; Burrus, C.S. : On computing the discrete Hartley transform. IEEE Trans. Acoust., Speech and Signal Processing. ASSP–35 (1985) 1231–1238

4.24 Meckelburg, H.J. ; Lipka, D. : Fast Hartley transform algorithm. Electronics Letters. 21 (1985) 341–343

4.25 Hsu, C.Y. ; Wu, J.L. : Fast computation of discrete Hartley transform via Walsh–Hadamard transform. Electronics Letters. 23 (1987) 466–468

4.26 Hsu, C.Y. ; Wu, J.L. : The Walsh–Hadamard/discrete Hartley transform. Int. J. Electronics. 62 (1987) 744–755

4.27 Buneman, O. : Conversion of FFT's to fast Hartley transforms. SIAM J. Sci. Stat. Comput. (1986) 624–639

4.28 Makhoul, J. : A fast cosine transformation in one and two dimensions. IEEE Trans. Acoust., Speech and Signal Processing. ASSP–28 (1980) 27–34

4.29 Ahmed, N. ; Natarajan, T. ; Rao, K.R. : Discrete cosine transform. IEEE Trans. Comput. C–23 (1974) 90–93

4.30 Narasimha, M.J. ; Peterson, A.M. : On the computation of the discrete cosine transform. IEEE Trans. Commun. COM–26 (1978) 934–936

4.31 Malvar, H.S. : Fast computation of discrete cosine transform through fast Hartley transform. Electronics Letters. 22 (1986) 352–353
— : Fast Computation of the discrete cosine transform and the discrete Hartley transform. IEEE Trans. Acoust., Speech and Signal Processing. ASSP–35 (1987) 1484–1485

4.32 Yip, P. ; Rao, K.R. : Sparse–matrix factorization of discrete sine transform. Proc. 12th Annu. Asilomar Conf. Circuit, Syst., Comput. 608 (1978) 549–555

4.33 Rao, K.R. ; Yip, P. : The discrete cosine transform. New York: Academic Press 1990

4.34 Lee, B.G. : A new algorithm to compute the discrete cosine transform. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-32 (1984) 1243-1245

4.35 Kanchan, V.R. ; Rao, K.R. : Modified discrete cosine transform. Proc. IEEE Region V Conf. (1983) 174-179

4.36 Roeser, P.R. ; Jernigan, M.E. : Fast Haar transform algorithms. IEEE Trans. Comput. C-31 (1982) 175-177

4.37 Wang, Z.D. : New algorithm for the slant transform. IEEE Trans. Pattern Anal. Mach. Intell. PAMI-4 (1982) 551-555

4.38 Fino, B.J. ; Algazi, V.R. : Slant-Haar transform. Proc. IEEE 62 (1974) 653-654

Kapitel 5

5.1 Arazi, B. : Two-dimensional digital processing of one-dimensional signal. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-22 (1974) 81-86

5.2 Mersereau, R.M. ; Speake, T.C. : A unified treatment of Cooley-Tukey algorithms for the evaluation of the multidimensonal DFT. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-29 (1981) 1011-1018

5.3 Eklundh, J.D. : A fast computer method for matrix transposing. IEEE Trans. Comput. C-21 (1972) 801-803

5.4 Yeung, K. ; Rath, D. ; Rao, K.R. : Generation of the fast 2D discrete Fourier transform. In: T.S. Young, et al. (Ed.): Signal processing III: theories and applications. Amsterdam: Elseviers Science Publ. (North Holland) (1986) 1067-1070

5.5 Rivard, G.E. : Direct fast Fourier transform of bivariate functions. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-25 (1977) 250-252

5.6 Harris, D.B. ; McClellan, J.H. ; Chan, D.S.K. : Schuessler H.W.: Vector radix fast Fourier transform. Proc. IEEE Internat. Conf. Acoust., Speech, Signal Processing. (1977) 548-551

5.7 Naccarato, D.F. ; Chien, Y.T. : A direct two-dimensional FFT with application in image processing. Proc. IEEE Conf. on Pattern Recognition and Image Processing. (1979) 233-238

5.8 Hoyer, E.A. ; Berry, W.R. : An algorithm for the two-dimensional FFT. Proc. IEEE Int. Conf. Acoust., Speech and Signal Processing. (1977) 552-555

5.9 Carlsohn, M. : Datenreduktion bei der digitalen Übertragung von Bildsignalen durch lokal veränderliche Auflösung. VDI Fortschrittberichte, Reihe 10 Nr.64. Düsseldorf: VDI-Verlag 1987

Kapitel 6

6.1 Guessom, A. : Fast algorithms for the multidimensonal discrete Fourier transform, Ph.D. Thesis Georgia Institute of Technology, School of Electrical Engineering 1984

6.2 Mou, Z.J. ; Duhamel, P. : In-place butterfly style FFT of 2-D real sequences. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-36 (1988) 1642-1650

6.3 Bracewell, R.N. ; Buneman, O. ; Hao, H.; Villasenor, J. : Fast two-dimensional Hartley transform. Proc. IEEE 74 (1986) 1282-1283

6.4 Kumaresan, R. ; Gupta, P.K. : Vector radix algorithm for a 2-D discrete Hartley transform. Proc. IEEE 74 (1986) 755-757

6.5 Haque, M.A. : A two-dimensional fast cosine transform. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-33 (1985) 1532-1539

Kapitel 7

7.1 Jesshope, C.R. : The implementation of fast radix-2 transforms on array processors. IEEE Trans. Comput. C-29 (1980) 20-27

7.2 Papamichalis, P.E : FFT implementation on the TMS320C30. Proc. IEEE Int. Conf. Acoust., Speech and Signal Processing. (1988)

7.3 Li, Z. ; Sorensen, H.V. ; Burrus, C.S. : FFT and convolution algorithms on DSP microprocessors. Proc. IEEE Int. Conf. Acoust., Speech and Signal Processing. (1986) 289-291

7.4 Besslich, Ph. W. ; Kurowski, J.O : Quadrant architecture for fast in-place algorithms. IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management (1983) 26-33

7.5 Besslich, Ph. W. : Parallel architecture for line-scanned images. In : Duff, M.J.B. ; Siegel, H.J. Corbett, F.J. (Eds.) : Architectures and Algorithms for Digital Image Processing. SPIE Proc. 596 (1985) 27-35

7.6 Besslich, Ph. W. : Pyramidal transforms in image processing and computer vision. In : Cantoni, V. ; Levialdi, S. (Eds.) : Pyramidal systems for computer vision. Berlin, Heidelberg, New York, London, Paris, Tokyo : Springer 1986, 215-246

7.7 Norton, A. ; Silberger, A.J. : Parallelization and performance analysis of the Cooly-Tukey-FFT algorithm for shared-memory architectures. IEEE Trans. Comput. C-36 (1987) 581-590

7.8 Dasgupta, S. : Computer Architecture, Vol. 2 : Advanced topics. New York : J. Wiley & Sons 1989

7.9 Siomalas, K. O. ; Bowen, B. A. : Synthesis of efficient pipelined architectures for implementing DSP operations. IEEE Trans. Acoust., Speech and Signal Processing. ASSP-30 (1985) 1499-1508

7.10 Shyu, H. C. ; Troung, T. K. ; Reed, I. S. ; Hsu, I. S. : Pipeline prime-factor DFT for VLSI using cyclic shuffling. IEE Proc. 134 Pt.E (1987) 247-253

7.11 Wold, E. H. ; Despain, A. M. : Pipeline and parallel-pipeline FFT processors for VLSI implementations. IEEE Trans. Comput. C-33 (1984) 414-425

7.12 Liu, W. : A New pipelined/parallel architecture for two dimensional fast Fourier transform. Proc. IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management (1983) 214-221

7.13 Yeh, H.-G. ; Yeh, H.-Y. : Implementation of the discrete Fourier transform on 2-dimensional systolic processors. IEE Proc. 134 Pt.G (1987) 181-185

7.14 Gertner, I. ; Shamash, M. : VLSI architectures for multidimensional Fourier transform processing. IEEE Trans. Comput. C-36 (1987) 1265-1274

7.15 Vetterli, M. ; Ligtenberg, A. : A discrete Fourier cosine transform chip. IEEE J. on Selected Areas in Commun. SAC-4 (1986) 49-61

Sachverzeichnis

Abtastung 2
Anwendung 2ff.
Array von Prozessoren 297

Basisbilder 3,204f.,218ff.
Basisfunktionen 2ff.,14,20,25,
27,29,36,41
Berechnung aus Koeffizienten
von Teilfolgen (s.
Koeffizienten aus Teilfolgen)
Berechnungsstrategien 43f.,204ff.
Binärdarstellungen 83
Bitaustauschoperationen 65ff.,209
Bitumkehroperation 26ff.,65,82,84,
88,219ff.

Cal/Sal-Ordnung (der WHT) 26f.,33
chinesisches Rest-Theorem 45ff.
Chrestenson-Funktionen 21
Cooley-Tukey-Algorithmus 78f.,80ff.,
256ff.
Cosinus-Transformation 19f.

DCT-Algorithmen 153 (s.a.:
direkte DCT, Koeffizienten
aus Teilfolgen, 2D-DCT)
DCT-Algorithmus mit
Matrixfaktorisierung 167ff.
DCT-Algorithmus
-mittels 2N-DFT 154ff.
-mittels N-DFT 159ff.
DFT (s.a.: FFT) 12ff.
DHYT-Algorithmen 130 (s.a.:
Primfaktor-Algorithmus,
Split-Radix-Algorithmus,
Koeffizienten aus Teilfolgen,
Koeffizientenkonvertierung,
2D-DHYT)
DHYT-Algorithmus
-mit Frequenzdezimierung 134ff.
-mit Zeitdezimierung 131ff.
Diagonalmatrix 6,50f.
Dimension 4,204
direkte Berechnung (2D) 215
direkte Koeffizienten-
berechnung 43
direkter 2D-DCT-Algorithmus 279ff.
direkter 2D-FFT-Algorithmus 2,241ff.
direkter 2D-WHT-Algorithmus 218
direkter DCT-Algorithmus 174ff.
direktes Produkt 51
diskrete Cosinus-Transformation 19f.
diskrete Fourier-Transformation 12ff.
diskrete Korrelation 15
diskrete Haar-Transformation 34ff.
diskrete Hartley-Transformation 17f.

diskrete Slant-Transformation 37ff.
diskrete Walsh-Transformation 21ff.
diskrete Sinus-Transformation 20
Drehfaktor 12,90f.,263,294
DST über DCT-Algorithmus 191f.
DST-Algorithmus mit
Matrixfaktorisierung 187ff.
Dualzahldarstellung (der Indizes)
23,80
dyadische Ordnung (der WHT) 31f.

Eigenschaften der 1D-FT 14f.
Eigenschaften der 2D-FT 232
Eigenschaften der DHYT 18
Eigenschaften der Walsh-
Funktionen 22
Einheitsmatrix 6,50

FFT-Algorithmus (s.a.:
Cooley-Tukey-Algorithmus,
Sande-Tukey-Algorithmus,
Radix-4-FFT-Algorithmus,
Split-Radix-Algorithmus,
Primfaktor-Algorithmus,
Winograd-Algorithmus,
Koeffizientenkonvertierung,
Koeffizienten aus Teilfolgen,
2D-FFT,
Vektor-Radix-Algorithmus,
Walsh-FT-Algorithmus)
FFT-Algorithmus
-für reelle 1D-Daten 120ff.
-für reelle 2D-Daten 266ff.
-für reelle und (anti-)
symmetrische Daten 120ff.
-mit Frequenzdezimierung 86f.
-mit Zeitdezimierung 85f.
-nach Cooley und Tukey 80ff.
Fourier-Basisfunktionen 14
Frequenz 1f.
Frequenzdezimierung (DIF) 86ff.
Frequenzspektrum 3
FT reeller Daten 16

Gray-Code 24f.
Grundschmetterling 61f.,294

Haar-Matrizen 36f.
Haar-Basisfunktionen 36
Haar-Transformation 34ff.
HT-Algorithmus 193ff.
HT-Algorithmus (in-place) 196f.
Hadamard-Matrix 32,52ff.
Hadamard-Ordnung (der WHT) 32
Hardware-Realisierung 296ff.
hermitische Matrix 50
hierarchische Transformation 207
hierarchische 2D-WHT 218f.
hierarchische Berechnung 207

Implementierung
-auf Signalprozessoren 293
-für komplexe Werte 294
Indizierung 57ff.
"in-place"-Verarbeitung 60ff.,293
Iteration 61

Koeffizientenberechnung
aus Teilfolgen 43f.,221f.
-für DCT 178ff.
-für 2D-DCT 289ff.
-für DFT 113ff.

-für 2D-DFT 250ff.
-für DHYT 141ff.
-für WHT 77ff.
Koeffizientenkonvertierung 44f.
-DCT---->DST 191f.
-DHYT---->DCT 165f.
-DHYT---->DFT 151f.
-WHT---->DFT 107ff.
-WHT---->SLT 198ff.
-HT---->SLT 203
Komplexität 9f.,88f.
Koordinantentransformation 4
Kronecker-Potenz 54ff.
Kronecker-Produkt 51ff.,57

Matrixfaktorisierung 50ff.
Matrixnotation 6,50
modifizierte DCT (MDCT) 182
Module 57ff,101ff.

natürliche Ordnung (der WHT) 32

Objektbereich 1
orthogonale Matrix 6
orthogonale Transformationen 6f.
orthogonale Vektoren 6
orthogonale Systeme 7
orthonormale Vektoren 6
Ortsbereich 1
Ortsfrequenz 2

Paley-Ordnung (der WHT) 31
parallele Verarbeitung 295
Parsevalsche Gleichung 7
Periodizitätsmatrix 257ff.
Permutationsmatrix 50,56,146,190
physikalische Dimension 1,4
Pipeline-Architektur 296
Polynom-Transformation 16f.
Primfaktor-DHYT-Algorithmus 136ff.
Primfaktor-FFT-Algorithmus 97ff.

Quantisierung 2
Quasidiagonalmatrix 51

Radix-2-Signalflußgraph 61f.
Radix-4-FFT-Algorithmus 62f.
Radix-p-Transformation 62
Rechenzeit 9,295
rekursive Koeffizienten-
 berechnung (s.:
 Koeffizientenberechnung
 aus Teilfolgen)
Restklassenarithmetik 45ff.

Sande-Tukey-Algorithmus 80,86ff.
Schmetterling 61ff.
Separierbarkeit des
 Transformationskerns 211ff.
Sequenz 2
Sequenz-Ordnung (der WHT) 28f.
serielle Verarbeitung 297
Signalflußgraph 60ff.
Sinus-Basisfunktion 20
Sinus-Transformation 19f.
ST-Algorithmus 187ff.
-mit Matrixfaktorisierung 187ff.
-über DCT 191f.
Slant-Transformation 37ff.
SLT-Algorithmus 198
-über HT 203
-über WHT 198ff.

-mit Faktorisierung 198
Software-Implementierung 293ff.
Split-Radix-DHYT-Algorithmus 138ff.
Split-Radix-FFT-Algorithmus 90ff.
Stützstellen 2,8f.,263
Summenform 8
symmetrische Matrix 50
systolisches Array 298

Transformationsbereich 1
Transformationskern 7,211
Twiddle-Faktor 99

unitäre Matrix 6
unitäre Transformation 5f.

Vektor-Radix-Algorithmus
-für DFT 235ff.
-für die DHYT 273ff.
verallgemeinertes
Kronecker-Produkt 56f.
vereinheitlichter
2D-DFT-Algorithmus 256ff.
Vollständigkeit 7
Vollständigkeitsrelation 7

Walsh-Basisfunktionen 22ff.
Walsh-Fourier-Transformations-
Algorithmus 107ff.
Walsh-Hadamard-Transformation 21ff.
Walsh-Ordnung (der WHT) 28,30
Walsh-Transformation
-in Cal/Sal-Ordnung 33
-in Hadamard-Ordnung 32
-in Paley-Ordnung 31
-in Walsh-Ordnung 28,30
WHT-Algorithmus
-in Hadamard-Ordnung 75f.
-in Paley-Ordnung 75f.
-in Walsh-Ordnung 74f.
Winograd-FFT-Algorithmus 103

Zusammenhang zwischen
DFT und DHYT 151
zahlentheoretische Trans-
formation 17
Zeile/Spalte-Algorithmus 211ff.
-der DFT 233f.
-der DHYT 271f.
Zeitdezimierung (DIT) 85f.
zelluläre Prozessoren 298ff.

2D-DCT-Algorithmen 277f.
2D-DFT-Algorithmen 225ff.
2D-DHYT-Algorithmen 270ff.
2D-Transformation mit
1D-Algorithmus 208ff.
2D-WHT-Algorithmen 225ff.