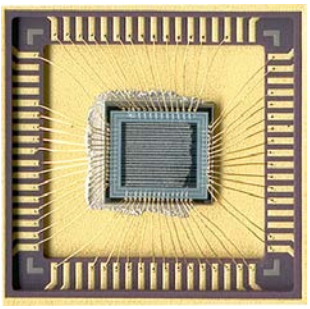
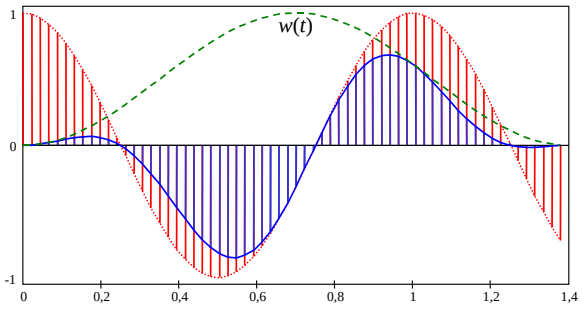
Karlsruher Institut für Technologie

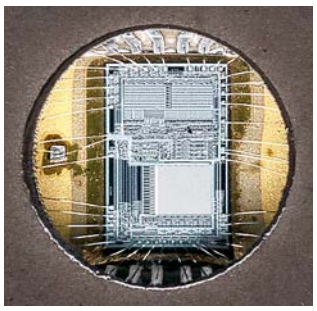
Institut für Industrielle Informationstechnik

Hertzstr. 16, Geb. 06.35

Prof. Dr.-Ing. habil. Klaus Dostert



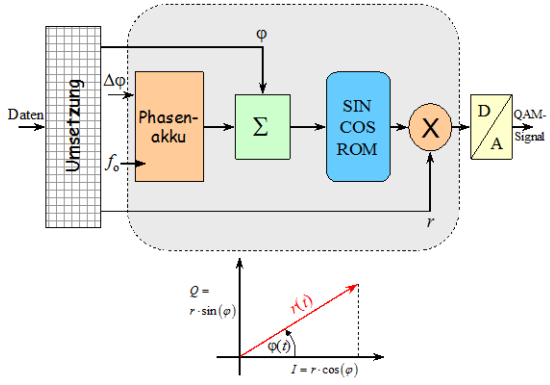


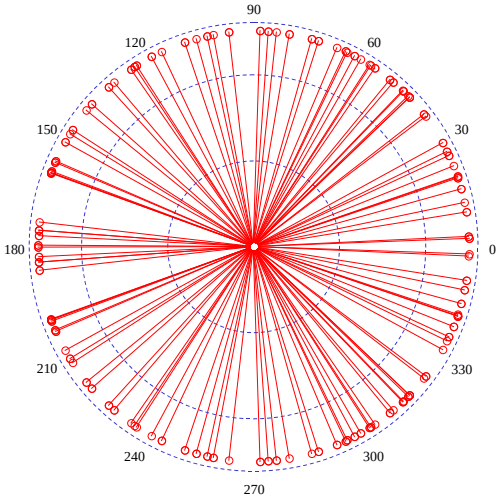


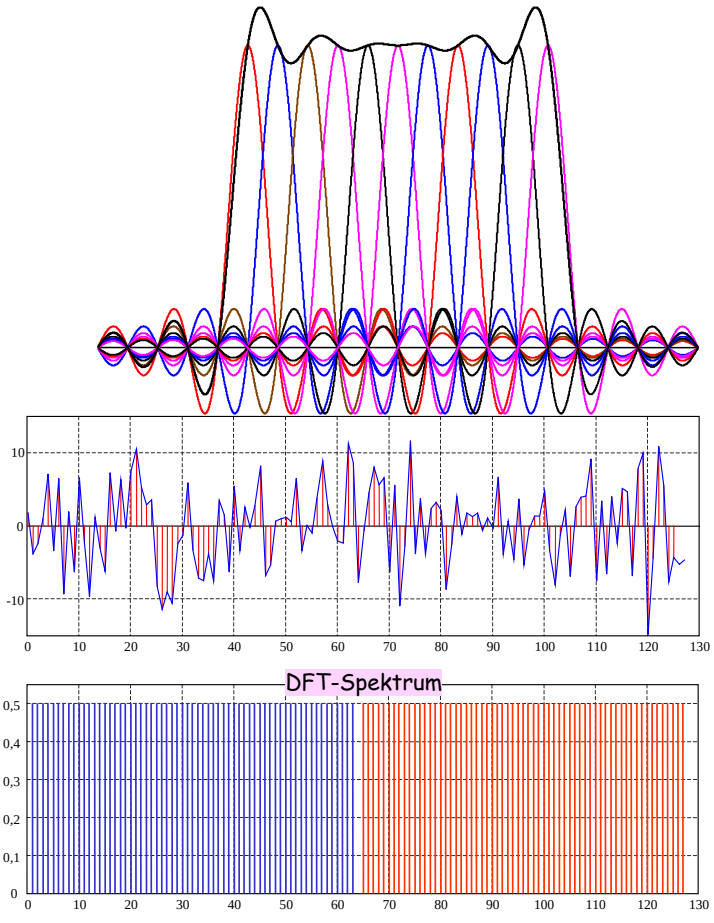
Skriptum zur Vorlesung

Integrierte Signalverarbeitungssysteme

Wintersemester 2015/2016







Themengebiete

Alle Unterlagen zu ISVS unter: **http://www.iiit.kit.edu/isvs.php**

Elemente der echtzeitfähigen Signalerzeugung und -verarbeitung

- Analog- und Digitalkomponenten
- Arithmetik, Software, Protokolle
- RISC-Strukturen
- spezielle Speicher- und Bussysteme (Harvard-Architektur)
- Interrupt–Verarbeitungskonzepte
- Timersysteme, Pulsweitenmodulation

Algorithmen und zugehörige besondere Hardwarestrukturen der DSV

- diskrete Faltung, Korrelation, Filterung, DFT
- Parallelmultiplizierer, Quadrierer, MAC–Einheiten
- systolische Pipelines
- Barrel-Shifter
- Zero-Overhead-Looping

Spezielle Funktionseinheiten digitaler Signalverarbeitungssysteme

- Signalsynthese: Wavetable- und AWG-Konzepte, direkte Digitale Synthese (DDS, NCO)
- Digitale Mischer
- Modulatoren und Demodulatoren (Software-Radio-Konzepte)
- FFT/IFFT–Prozessoren
- Echo–Entzerrer
- FIR-, IIR- und Kammfilter, adaptive Filter

VHDL-Entwurf hochintegrierter Signalverarbeitungsschaltungen

- System-, Algorithmen-, Register-Transfer-, Gatter-, Schalter-, Layoutebene
- Entwurfsstile und Methodik des hierarchischen Entwerfens
- Eigenschaften und Einsatz von FPGAs, Gate-Arrays, Standardzellen, Full-Custom ICs
- Entwicklungsumgebung: Altera-Quartus
- Modellerstellung, Simulation, Synthese, Placement & Routing, Verifikation, Test
- Mixed-Signal-Entwurf (monolithische Integration analoger und digitaler Funktionseinheiten) Besonderheiten analoger Schaltungsfunktionen: AGC, Vorfilterung, A/D- und D/A-Wandlung, Sendeverstärker, Signalkoppler

Anwendungsbeispiele

- Multicarrier (OFDM) Systeme in der Telekommunikation (xDSL, DAB, DVBT, RDM, PLC)
- Universelle ’Channel-Sounder’ (Kanalvermessungssysteme)
- Kanalemulatoren z.B. für PLC-Kanäle und ISM-Funkbänder

Integrierte Signalverarbeitungssysteme (ISVS)

(2V +1Ü ≡3SWS)

Inhaltsübersicht

1 Elemente der echtzeitfähigen Signalerzeugung und -verarbeitung

Die digitale Signalverarbeitung (DSV) hat die Entwicklung der Technik im letzten Drittel des 20. Jahrhunderts in atemberaubendem Umfang beeinflußt. So wird heute nicht nur Datenverarbeitung sondern auch Nachrichten- und Hochfrequenztechnik digital realisiert. Selbst in der Starkstromtechnik, der Leistungselektronik und der Kraftwerkstechnik sind digitale Steuerungen und Regelungen nicht mehr wegzudenken. Während die klassische Datenverarbeitung, auf die die ersten Mikroprozessoren abzielten, in den meisten Fällen nur 'offline', d.h. ohne feste und verläßliche Zeitvorgaben durchgeführt werden kann, stehen bei den heutigen Aufgaben der DSV in der Regel „harte" Echtzeitanforderungen im Vordergrund. Der herkömmliche Mikroprozessor oder Mikrocontroller ist für solche Aufgaben nur sehr eingeschränkt verwendbar.

Während die wesentlichen theoretischen Grundlagen schon in den 60-er Jahren zur Verfügung standen, bedurfte es noch enormer Fortschritte der Halbleitertechnologie, um die Algorithmen auf digitaler Hardware echtzeitfähig implementieren zu können. Einen wichtigen Meilenstein in dieser Richtung stellte Anfang der 80-er Jahre der digitale Signalprozessor (DSP) dar. Dieser Baustein grenzte sich sowohl in der Architektur als auch im Befehlssatz deutlich vom Mikroprozessor ab. Dies war aufgrund der Zielrichtung – nämlich harte Echtzeitanforderungen zu ermöglichen – unvermeidbar. Die Abgrenzung führte aber dazu, daß die ersten DSP enorm teuer in der Herstellung und aufwendig in der Programmierung waren, so daß auch der mit Mikroprozessoren vertraute Ingenieur erhebliche Hürden beim Umstieg vor sich sah. Die Einführung erfolgte deshalb zunächst sehr zögerlich, und es dauerte fast bis zum Ende des letzten Jahrhunderts, bis der DSP seinen Siegeszug in alle Gebiete der Technik in voller Breite erreichte.

Obwohl der DSP eine Art Krönung der Digitaltechnik darstellt, ist die Entwicklung mit ihm keineswegs zu einem Abschluß gekommen. Mit fortschreitender Integration technischer Systeme stellte sich die universelle Verwendbarkeit des DSP, auf die er konzipiert ist, als massives Hindernis bei der Systemoptimierung heraus. Während ein schneller, leistungsfähiger DSP zwar im Grunde alle denkbaren Aufgaben der DSV ausführen kann, „schleppt" er in der speziellen Anwendung häufig eine Menge Redundanz mit, die nicht benötigt wird. Auf der anderen Seite – das wird uns in der Kommunikationstechnik oft begegnen – müssen oft mehrere Funktionen parallel zur Verfügung stehen, wovon jede für sich betrachtet zwar relativ einfach ist, die zusammen aber dennoch rasch die Ressourcen des DSP aufbrauchen. Digitale Filter sind ein gutes Beispiel dafür. Es ist sofort einzusehen, daß man in solchen Fällen mit einem anwendungsspezifischen Ansatz, der nur die benötigten Bestandteile eines DSP zusammenfügt und diese parallel laufen läßt, erheblich übersichtlichere und kostengünstigere Lösungen erhalten kann. Man kann den einzelnen DSP oder auch Mikroprozessor im System nicht mehr ausmachen, weil er in vielfältiger Weise integriert, d.h. eingebettet ist. Für den neuen Ansatz hat sich der Begriff '***Embedded System***' schon seit Jahren eingebürgert. Während sich anfangs eher ein Wunschtraum dahinter verbarg, ist man mit der heutigen Halbleitertechnik in der Lage, ***Embedded Systems*** ungeahnter Komplexität auf einem Siliziumchip monolithisch zu integrieren. Die aktuelle Herausforderung für den Ingenieur ist, solche Systeme in Entwurf, Verifikation und Test zu beherrschen.

Besonders die Kommunikationstechnik bietet eine enorme Vielfalt von Aufgaben. Zu den Dingen, die heute bereits digitale Systeme erledigen, zählen Filtern, Korrelieren, Transformieren, Kodieren/Dekodieren, Modulieren/Demodulieren Komprimieren/Dekomprimieren, Fehlerkorrektur und Signalsynthese. Es handelt sich dabei praktisch immer um Aufgaben mit „harten" Realzeitanforderungen, an denen sich zeigen läßt, daß der Weg zur anwendungsspezifischen Lösung deutliche Vorteile gegenüber programmierbaren Universalbausteinen bringt. Die Realisierung eigener, den jeweiligen technischen Problemstellungen angepaßten Lösungen in Form von 'Application Specific Integrated Circuits' (ASICs) ist kein langwieriger, kostenintensiver Prozeß mehr, denn ein komplexes ASIC, z.B. in Form eines FPGA (≡ Field Programmable Gate-Array), kann mit preisgünstiger PC-Ausstattung am Arbeitsplatz des Ingenieurs vollständig entwickelt und getestet werden.

Beispiele für Echtzeitaufgaben

- Filtern von Signalen
- Modulation, Demodulation
- Faltung (diskret)
- Korrelation
- Transformation (z.B. Quadrieren, Logarithmieren, Fouriertransformieren)
- Gleichrichtung, Verstärkung, Ent– oder Verzerrung
- Manipulationen: Frequenzverschiebung, zeitliche Verwürfelung (Vocoder, Scrambling)

Einige dieser Aufgaben sind auf analogem Wege nicht oder nur unter enormem Aufwand zu bewerkstelligen

Vergleich von Signalverarbeitungsmethoden

analog	digital
Komponentenselektion	stabile, deterministische Funktion
Abgleich in der Regel erforderlich	kein Abgleich nötig
temperaturabhängig	enge Toleranzen auf Dauer
Bauteilealterung	keine Veränderungen während der „normalen“ Systemlebensdauer
störempfindlich auch bei schwachen Störsignalen	unempfindlich für Störungen bis zu sehr hohen Schwellwerten
Modifikation aufwendig oder unmöglich	leicht modifizierbar, sogar ohne Betriebsunterbrechung
adaptive Funktionen kaum realisierbar	adaptive Funktionen sind einfach durch Software realisierbar
aufwendiges Testen	ständige Selbsttestmöglichkeiten

Typische Aufgaben der digitalen Signalverarbeitung

Filterung

- FIR[1]-Filter - Filter mit endlicher Impulsantwort
- IIR[2]-Filter – Filter mit (theoretisch) unbegrenzter Impulsantwort (rekursive Struktur)
- signalangepaßte Filter (Matched Filter, MF)
- adaptive Filter (variable Koeffizienten)

[1] Finite Impulse Response
[2] Infinite Impulse Response

spezielle direkte Signalbeeinflussung

- Dynamikkompression und –expansion, z.B. für Sprache (Mobilkommunikation)
- Mittelwertbildung
- Leistungs- und Energieberechnung
- Modulation/Demodulation (Amplitude, Phase, Frequenz)
- Spektralanalyse (FFT, DFT)

spezielle Datenverarbeitung

- Kryptographie
- Scrambling (zeitliche Verwürfelung)
- Codierung
- Decodierung (z.B. Viterbi-Decodierung)

Wichtige Einsatzgebiete von DSV-Systemen

Telekommunikation

- Modulation/Demodulation
- Rauschunterdrückung, Echoentzerrung
- Tonerzeugung (z.B. Wählton - DTMF)
- Sprachverschleierung (Vocoder)
- Hochgeschwindigkeits-Modems (xDSL-Techniken)
- Mobilfunk (GSM, UMTS)
- lokale Funksysteme (WLAN ≡ Wireless Local Area Network)
- Rundfunk: DAB ≡ Digital Audio Broadcasting, DVBT ≡ Digital Video Broadcasting Terrestrial, DRM ≡ Digital Radio Mondiale

Sprach-, Audio- und Videosignalverarbeitung

- Spracherkennung
- Sprachsignalsynthese
- Sprecher-Identifizierung
- Musik-Synthese (elektronische Musik)
- Kompression (Datenreduktion), z.B. MP3
- akustische Equalizer
- digitale HiFi-Verstärker
- Bilddatenreduktion (MPEG)
- Signalaufbereitung für hochauflösende Bildschirme

Regelungstechnik und Maschinenbau

- Robotik
- Motorsteuerung, Antriebstechnik
- Turbinenüberwachung (Vibrationsanalyse)

Medizintechnik

- Sonographen, z.B. mit Ultraschall
- Röntgengerätesteuerung und Signalauswertung, Computertomographie
- EKG-Signalauswertung
- EEG-Signalauswertung
- Kernspintomographie

Radar- und Sonarsignalverarbeitung

- Navigation
- Ozeanographie
- Verkehrsleitsysteme, GPS
- geologische Forschung (Seismik)

Computer - Bildverarbeitung - Graphik

- Grafikbeschleuniger, speziell für 3D-Graphik (DTP, CAE, Animation)
- Mustererkennung
- Bildkorrektur (Qualitätsverbesserung)
- Bilddatenreduktion
- Roboter-Vision

Meßtechnik

- Spektralanalyse
- Signalsynthese, Erzeugung spezieller Testsignale und Testmuster
- Transientenanalyse
- Zeit-/Frequenzanalyse
- automatische Meßwertaufnahme und –verarbeitung sowie Fernübertragung

1.1 Bausteine der digitalen Signalverarbeitung

1.1.1 Mikroprozessor und Mikrocontroller

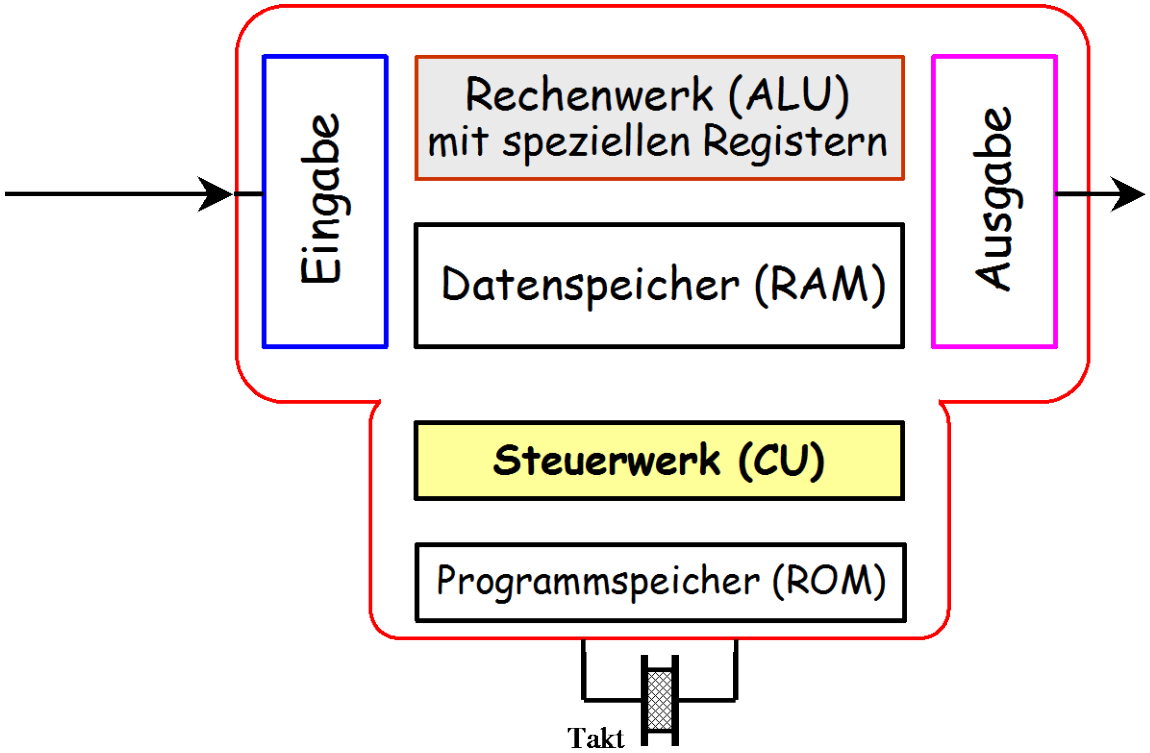


Bild 1.1: Grundstruktur eines Mikrorechners

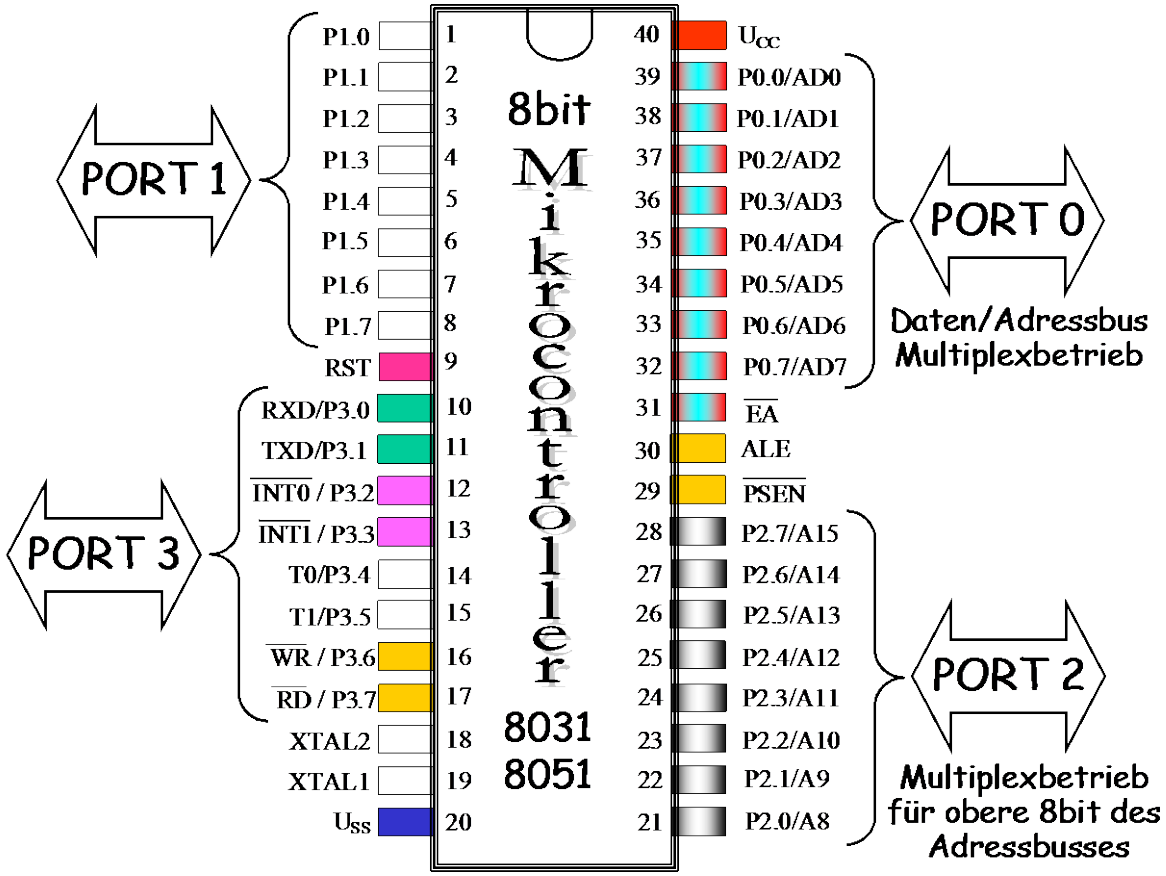


Bild 1.2: Pinbelegung eines 8bit-Mikrocontrollers in einem DIL-20-Gehäuse

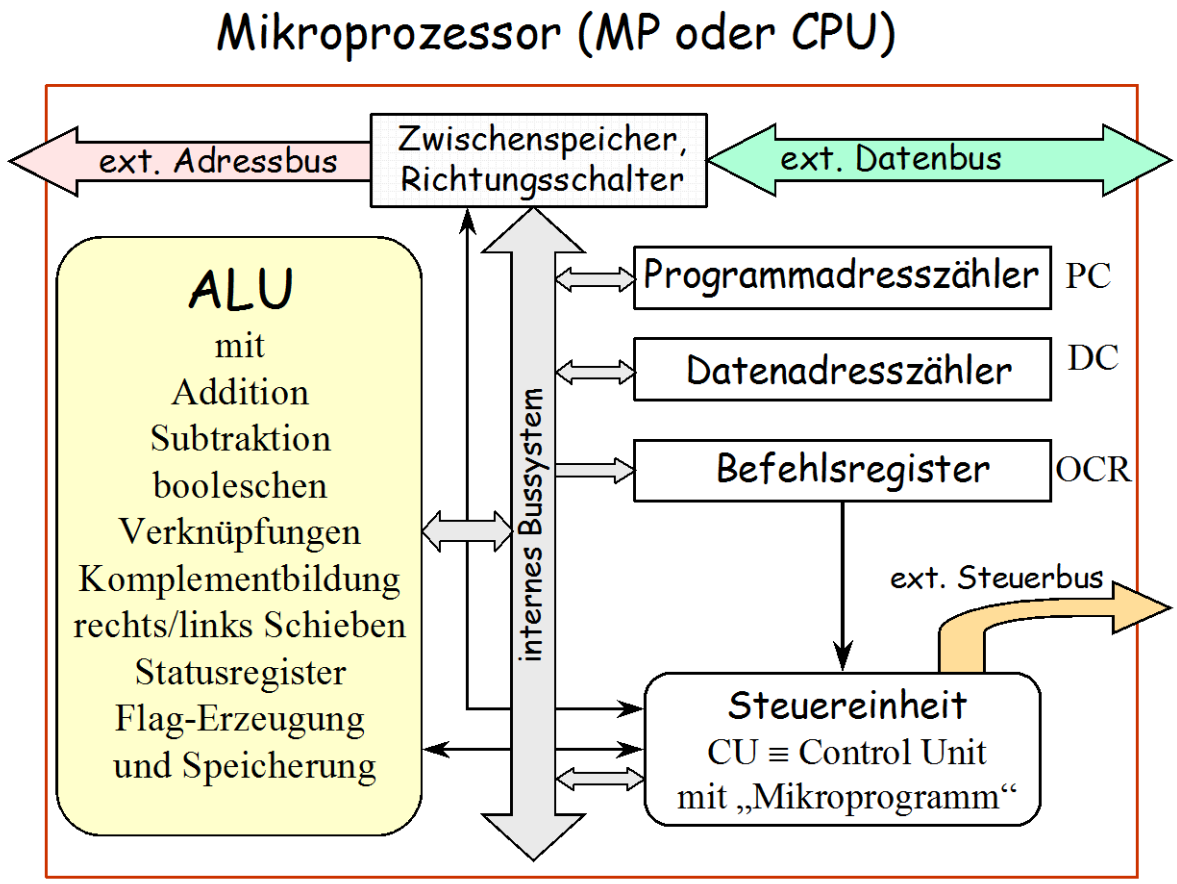


Bild 1.3: Blockschaltbild und Funktionen eines Mikroprozessors

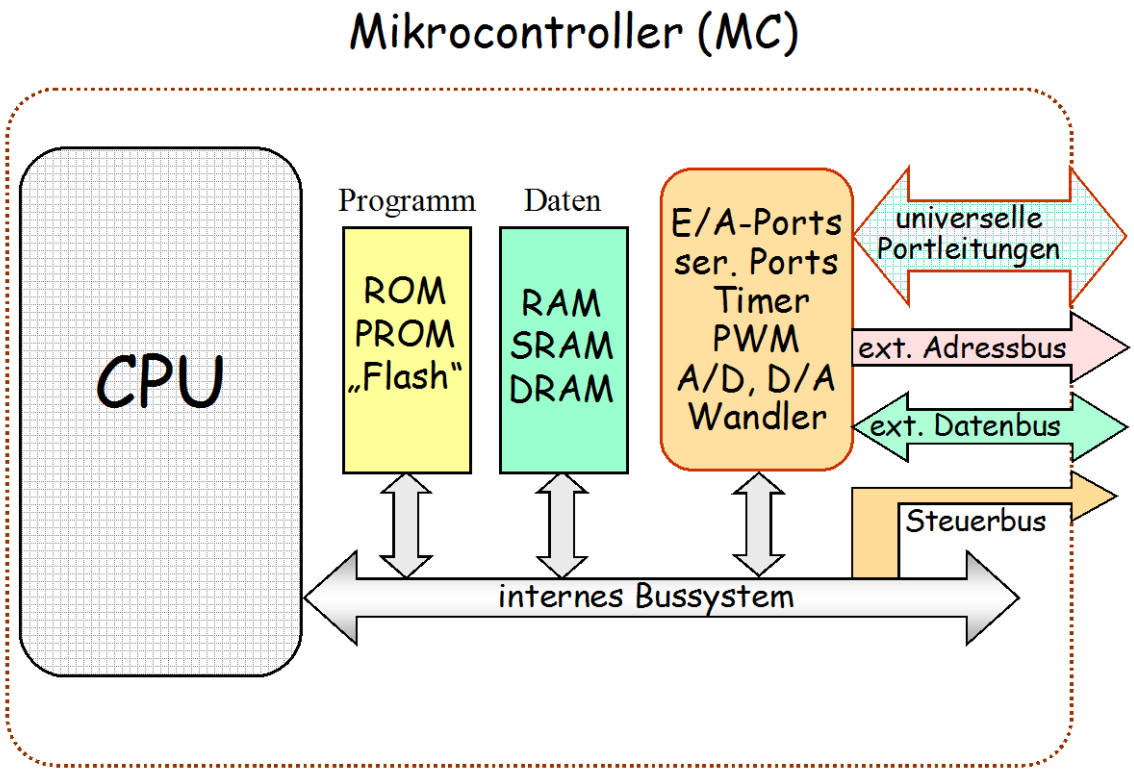


Bild 1.4: Der Mikrocontroller als vollständiges Rechnersystem

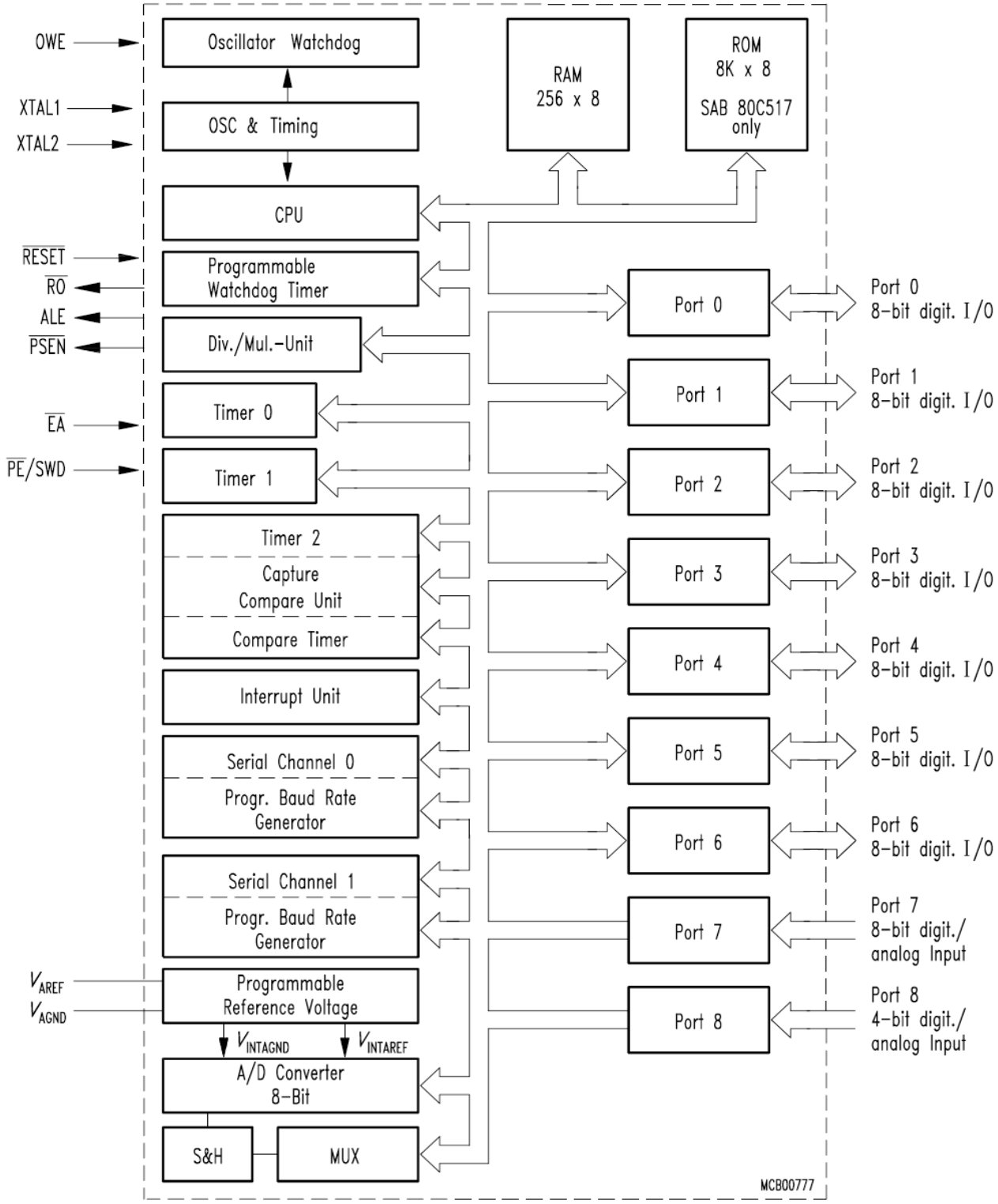


Bild 1.5: Blockschaltbild eines typischen ’High End’-8-bit-Mikrocontrollers (80517/537); ein mit zahlreichen Peripheriekomponenten erweiterter 8051-Kern

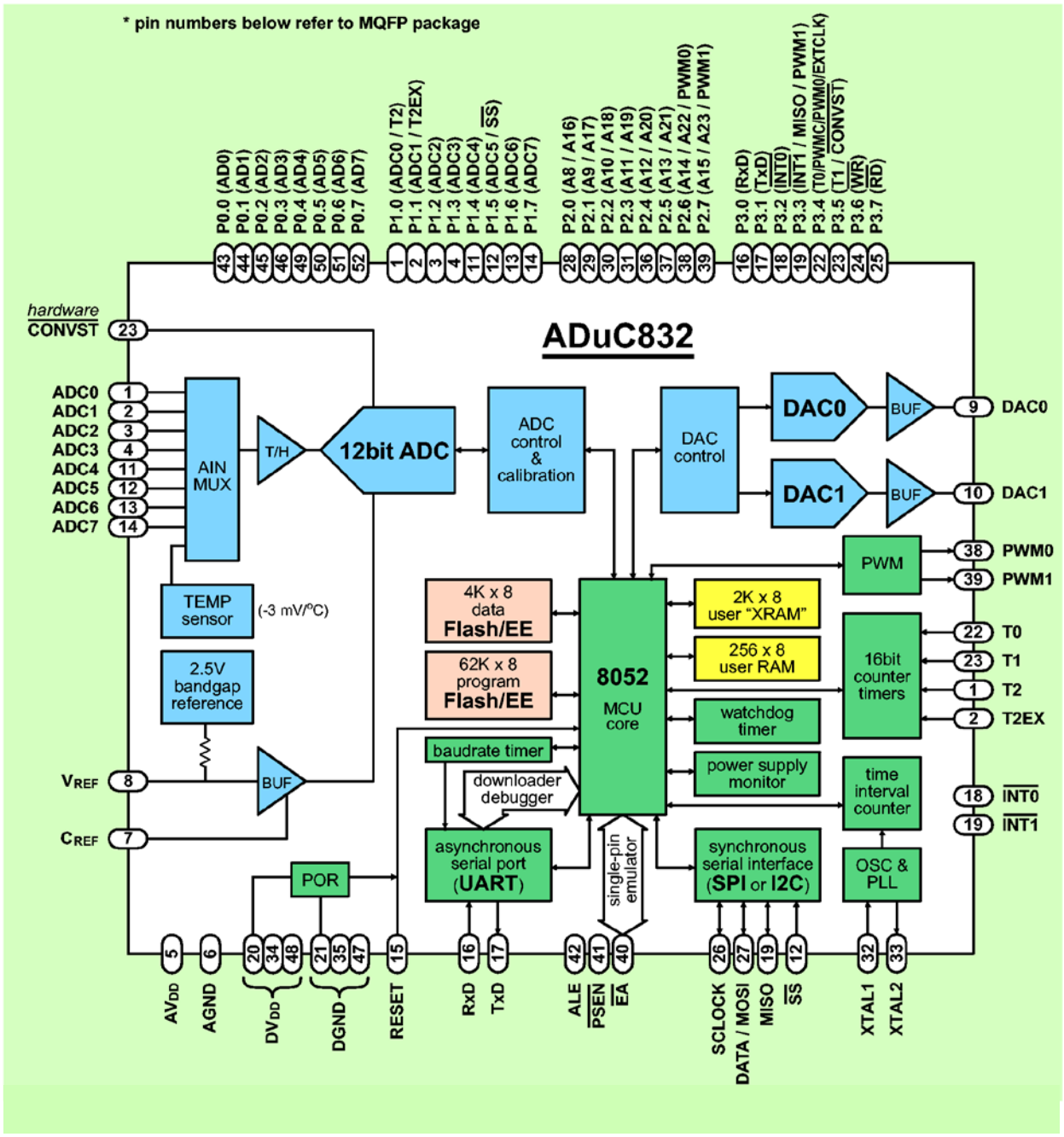


Bild 1.6: Das 8051-basierte Datenerfassungssystem ADuC832

1.1.2 Grenzen des Mikroprozessors bei Aufgaben der DSV

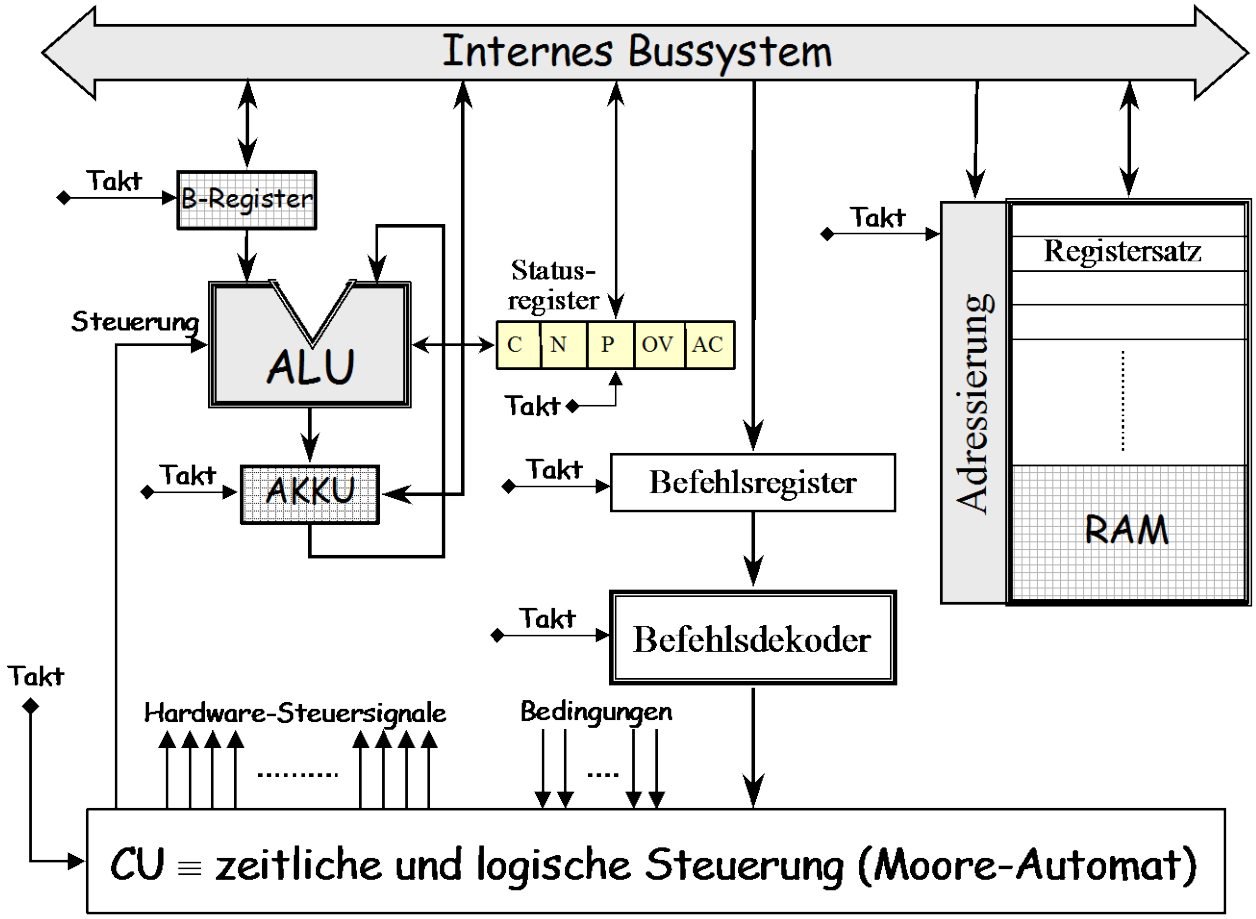


Bild 1.7: Der Standard-Mikroprozessor verfügt nur über Addierarithmetik und sehr beschränkte Datentransferressourcen

1.1.3 Arithmetische Basis der DSV: Die MAC-Operation

An dieser Stelle werden Beispiele von typischen Algorithmen der digitalen Signalverarbeitung angeführt, bei denen Multiplizier-Akkumulier-Operationen das Kernstück bilden. Es wird klar, daß die Anwendung von Mikroprozessoren und Mikrocontrollern hier – selbst bei sehr langsam veränderlichen Signalen an Grenzen stößt.

1. Diskrete Korrelation

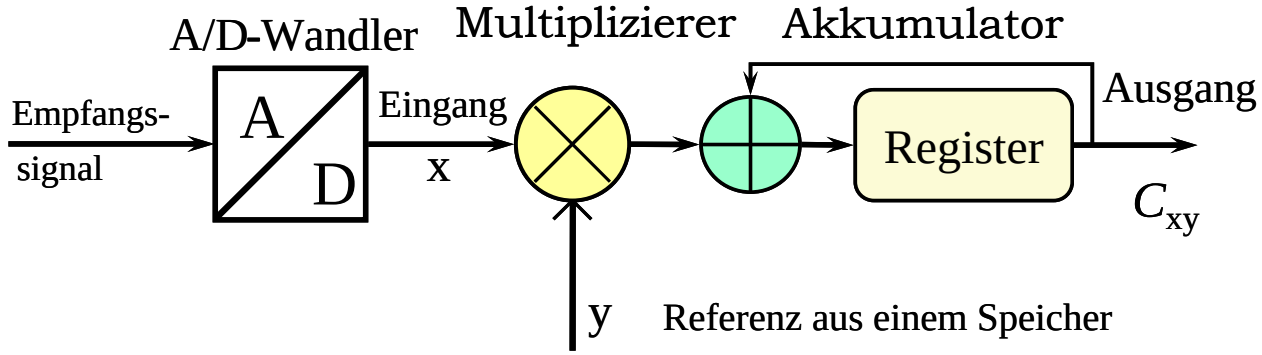


Bild 1.8: Grundaufbau eines digitalen Korrelationsempfängers

$$C_{xy}(k) = \sum_{i=0}^{N-1} x(i) \cdot y(k+i) \tag{1.1}$$

Im dem in Bild 1.8 dargestellten und mittels (1.1) beschriebenen Beispiel repräsentiert *x(k)* die digitalisierten Abtastwerte eines Empfangssignals, das mit der Abtastfrequenz $f_A=1/T$ abgetastet wurde, während die *y(k)* Referenzwerte aus einem Speicher sind. Es ist zur Vereinfachung der Schreibweise ist es üblich, die Größe *T* (Abtastzeitraster) wegzulassen und statt *(kT)* oder *(iT)* nur *(k)* bzw. *(i)* zu schreiben. Man muß aber darüber klar sein das *k* und *i* hier keine dimensionslosen Zählvariablen sind, sondern die Dimension „Zeit“ haben. Bei der diskreten Fouriertransformation wird analog bei der Darstellung des Spektrums verfahren. Dann muß man beachten, daß die Variable (z.B. *n*) nunmehr die Dimension Frequenz hat.

$$\begin{array}{lll} C'_{xy}(k) = x(\nu)\cdot y(\nu+k) & + & \sum_{i=0}^{\nu-1} x(i)\cdot y(i+k) \\ \updownarrow \qquad\qquad \updownarrow & & \updownarrow \\ S_\nu = \quad x\cdot y & + & S_{\nu\text{-}1} \end{array} \tag{1.2}$$

Anhand von (1.2) lassen sich die Schritte der MAC–Operation in einem Aufbau nach Bild 1.8 im Detail verfolgen. Der Akkumulator speichert jeweils die bislang aufgelaufene Summe $S_{\nu\text{-}1}$ zu der mit jedem Abtastwert ein neu berechnetes Produkt $X\cdot Y$ addiert wird. Nach *N* Abtastwerten ist das Korrelationsergebnis für eine Signalform der Dauer $N\cdot T$ komplett und kann in weiteren Funktionseinheiten eines digitalen Empfängers ausgewertet werden. In der Praxis wird die Bitbreite des Akkumulators deutlich größer als die des Produktes gewählt. Eine sinnvolle Konstellation bei einer A/D-Wandlerauflösung von 8bit (d.h. Produktbreite 16bit) ist z.B. eine Akkulänge von 32bit, so daß sich

$$S_\nu = -2^{31} + 2^8 + \sum_{i=15}^{30} 2^i + x_7 y_7 \cdot 2^{14} + \sum_{i=0}^{6}\sum_{j=0}^{6} x_i y_j \cdot 2^{i+j} + \sum_{i=0}^{6} \overline{x_7 y_i} \cdot 2^{7+i} + \sum_{i=0}^{6} \overline{y_7 x_i} \cdot 2^{7+i} - s_{(\nu-1)31}\cdot 2^{31} + \sum_{i=0}^{30} s_{(\nu-1)i}\cdot 2^i$$

ergibt.

2. <u>Diskrete Faltung</u>

$$F_{xy}(k) = \sum_{i=0}^{N-1} x(i)\cdot y(k-i) \tag{1.3}$$

3. <u>Diskrete Fouriertransformation (DFT)</u>

$$G\left(\frac{n}{NT}\right) = \sum_{i=0}^{N-1} g(kT)\cdot e^{-j2\pi nk/N}, \quad \text{mit } n = 0,1...N-1 \tag{1.4}$$

4. <u>Inverse diskrete Fouriertransformation (IDFT)</u>

$$g(kT) = \frac{1}{N}\sum_{n=0}^{N-1} G\left(\frac{n}{NT}\right)\cdot e^{j2\pi nk/N}, \text{ mit } k = 0,1,...N-1 \tag{1.5}$$

Die Algorithmen der DFT und der IDFT veranschaulichen in besondere Weise die Bedeutung der MAC–Operation. Die Argumente *(kT)* und *(n/NT)* sind vollständig ausgeschrieben, um klarzumachen, daß es sich bei *k* und *n* in der vereinfachten Schreibweise nicht mehr um reine Zahlen sondern um dimensionsbehaftete Größen, nämlich Zeit bzw. Frequenz handelt.

Schnelle Fouriertransformation (engl.: Fast Fourier Transform ⇒ FFT)

Ausgehend von (1.4) wird ein Eingangsvektor *g(k)* der Länge *N* im Zeitbereich angesetzt, was einem Signalausschnitt der Dauer $N\cdot T$ entspricht. Als Ergebnis erhält man ebenfalls einen Vektor

der Länge N im Frequenzbereich, der die Spektrallinien im Abstand $1/NT$ darstellt. Insgesamt wird also ein Spektrum der Breite $f_A=1/T$ beschrieben.

$$G(n)=\sum_{k=0}^{N-1} g(k)\cdot e^{-j2\pi nk/N}, \quad \text{mit } n=0,1...N-1 \tag{1.6}$$

Man sieht, daß (1.6) komplex ist, so das jede Multiplikation in Wirklichkeit zweimal, nämlich für Real- und Imaginärteil auszuführen ist. Führt man die Abkürzung

$W=e^{-j\frac{2\pi}{N}}$ ein, dann sind darin alle festen Größen enthalten und aus (1.6) wird

$$G(n)=\sum_{k=0}^{N-1} g(k)\cdot W^{nk}, \quad \text{mit } n=0,1...N-1 \tag{1.7}$$

Die Größe W^{nk} wird auch Drehfaktor (engl.: twiddle factor) genannt und spielt in der Abarbeitung von FFT-Algorithmen eine wichtige Rolle. Für die Berechnung wird W^{nk} in Real- und Imaginärteil zerlegt.

$$W^{nk}=e^{-j2\pi\frac{nk}{N}}=\cos\left(2\pi\frac{nk}{N}\right)-j\sin\left(2\pi\frac{nk}{N}\right), \quad \text{mit} \quad k,n\in\{0,1,...N-1\} \tag{1.8}$$

Man sieht, daß auch bei reellen Zeitfunktionen die DFT in der Regel komplex ist. Die komplette direkte Ausrechnung von (1.7) bedeutet somit eine zweifache NxN–Matrix–Multiplikation, d.h. es fallen $2N^2$ Multiplikationen an. Der besondere Vorteil von FFT-Algorithmen besteht darin, die Zahl der Multiplikationen durch geschickte Ausnutzung von Redundanzen erheblich zu reduzieren, nämlich auf $1/2\cdot N\cdot \mathrm{ld}(N)$. Bei den gebräuchlichsten FFT-Algorithmen ist N eine Zweierpotenz, so daß sich z.B. bei $N=2^{10}=1024$ eine Reduktion von $2^{21}=2.097.152$ Multiplikationen auf $2^{10}\cdot 10=10.240$ ergibt.

Neben der Multiplikation wird in der DSV, insbesondere in Kommunikationssystemen auch gelegentlich das Quadrieren benötigt, z.B. bei der geometrischen Addition $z=\sqrt{x^2+y^2}$. Um das Berechnen der Wurzel kommt man in der Regel herum, weil die erforderlichen Entscheidungen auch mit den quadrierten Daten ohne Einschränkung getroffen werden können.

Zum Vergleich von Multiplikation und Quadrieren wird zunächst das Produkt von zwei N-stelligen Binärzahlen A und B in Zweierkomplementdarstellung betrachtet.

$$\begin{aligned} A\cdot B &= \left(-a_{N-1}\cdot 2^{N-1}+\sum_{i=0}^{N-2} a_i 2^i\right)\cdot\left(-b_{N-1}\cdot 2^{N-1}+\sum_{i=0}^{N-2} b_i 2^i\right)= \\ &= a_{N-1}\cdot b_{N-1}\cdot 2^{2N-2}+\sum_{j=0}^{N-2}\sum_{i=0}^{N-2} a_i\cdot b_j\cdot 2^{i+j}-a_{N-1}\cdot 2^{N-1}\cdot\sum_{i=0}^{N-2} b_i 2^i-b_{N-1}\cdot 2^{N-1}\cdot\sum_{i=0}^{N-2} a_i 2^i \end{aligned} \tag{1.9}$$

Entsprechend den Rechenregeln in Zweierkomplementdarstellung wird die Beziehung

$$\mathrm{A}+\overline{\mathrm{A}}=2^{\mathrm{N}}-1 \quad \Rightarrow \quad -\mathrm{A}=-2^{\mathrm{N}}+\overline{\mathrm{A}}+1$$

auf die negativen Mischterme

$$-a_{N-1}\cdot 2^{N-1}\cdot\sum_{i=0}^{N-2}b_i 2^i=-2^{N-1}\cdot\sum_{i=0}^{N-2}a_{N-1}b_i\cdot 2^i=2^{N-1}\cdot\left[-\sum_{i=0}^{N-2}a_{N-1}b_i\cdot 2^i\right]$$

und

$$-b_{N-1}\cdot 2^{N-1}\cdot\sum_{i=0}^{N-2}a_i 2^i=-2^{N-1}\cdot\sum_{i=0}^{N-2}b_{N-1}a_i\cdot 2^i=2^{N-1}\cdot\left[-\sum_{i=0}^{N-2}b_{N-1}a_i\cdot 2^i\right]$$

angewandt. Damit erhält man:

$$2^{N-1}\left[-2^{N-1}+\sum_{i=0}^{N-2}\overline{a_{N-1}b_i}\cdot 2^i+1\right]=-2^{2N-2}+\sum_{i=0}^{N-2}\overline{a_{N-1}b_i}\cdot 2^{i+N-1}+2^{N-1}$$

und

$$2^{N-1}\left[-2^{N-1}+\sum_{i=0}^{N-2}\overline{b_{N-1}a_i}\cdot 2^i+1\right]=-2^{2N-2}+\sum_{i=0}^{N-2}\overline{b_{N-1}a_i}\cdot 2^{i+N-1}+2^{N-1},$$

woraus sich die Summe

$$-2^{2N-1}+2^N+\sum_{i=0}^{N-2}\overline{a_{N-1}b_i}\cdot 2^{i+N-1}+\sum_{i=0}^{N-2}\overline{b_{N-1}a_i}\cdot 2^{i+N-1}$$

ergibt. Das gesamte Produkt stellt sich dann wie folgt dar:

$$A\cdot B=-2^{2N-1}+2^N+a_{N-1}\cdot b_{N-1}\cdot 2^{2N-2}+\sum_{j=0}^{N-2}\sum_{i=0}^{N-2}a_i\cdot b_j\cdot 2^{i+j}+$$
$$+\sum_{i=0}^{N-2}\overline{a_{N-1}b_i}\cdot 2^{i+N-1}+\sum_{i=0}^{N-2}\overline{b_{N-1}a_i}\cdot 2^{i+N-1} \qquad (1.10)$$

Die Produktbildung der Komponenten $a_i b_j$ erfolgt mittels UND- und der negierten $\overline{b_i\cdot a_j}$ mittels NAND–Gattern. Wenn man die einzelnen Binärprodukte nach ihrer Wertigkeit ordnet, entsteht ein Matrixschema, das z.B. mit zeilenweiser Addition schrittweise abgearbeitet werden kann. Da es bei den Multiplizier-Akkumulier-Operationen (MAC) in der DSV aber auf Geschwindigkeit ankommt, müssen Methoden der Parallelberechnung angewandt werden.

Beim Quadrieren verbleibt nur noch ein Operand, nämlich

$$A=-a_n\cdot 2^{N-1}+\sum_{i=0}^{N-2}a_i\cdot 2^i \qquad (1.11)$$

Mit Hilfe von (1.10) erhält man, indem dort $B=A$ gesetzt wird, und unter Beachtung der Tatsache, daß das boolesches Produkt gleicher Binärvariabeln $a_i\cdot a_i=a_i$ ist, das Ergebnis:

$$A^2=-2^{2N-1}+a_{N-1}\cdot 2^{2N-2}+2^N+\sum_{i=0}^{N-2}\overline{a_{N-1}\cdot a_i}\cdot 2^{i+N}+\sum_{i=0}^{N-2}\sum_{j=0}^{N-2}a_i\cdot a_j\cdot 2^{i+j} \qquad (1.12)$$

Für N=17 soll die Richtigkeit von (1.12) beispielhaft verdeutlicht werden:

$$A^2=-2^{33}+a_{16}\cdot 2^{32}+2^{17}+\sum_{i=0}^{15}\overline{a_{16}\cdot a_i}\cdot 2^{i+17}+\sum_{i=0}^{15}\sum_{j=0}^{15}a_i\cdot a_j\cdot 2^{i+j}$$

Man erhält stets korrekte positive 32-stellige Binärzahlen als Ergebnis, wenn der Übertrag in die Stelle 2^{34} ignoriert wird. Dazu betrachten wir zunächst positive Zahlen, d.h. a_{16}=0

$$A^2 = -2^{33} + 2^{17} + \sum_{i=0}^{15} 1 \cdot 2^{i+17} + \sum_{i=0}^{15}\sum_{j=0}^{15} a_i \cdot a_j \cdot 2^{i+j} = -2^{33} + 2^{17} + \left[2^{32} + 2^{31} + 2^{30} + ... + 2^{17}\right] + \sum_{i=0}^{15}\sum_{j=0}^{15} a_i \cdot a_j \cdot 2^{i+j}$$

$$= -2^{33} + 2^{33} + \sum_{i=0}^{15}\sum_{j=0}^{15} a_i \cdot a_j \cdot 2^{i+j} = \sum_{i=0}^{15}\sum_{j=0}^{15} a_i \cdot a_j \cdot 2^{i+j}$$

Man sieht, daß das Ergebnis immer positiv ist und sich aus der Doppelsumme über die Binärvariablen $a_i \cdot a_j$ ergibt.

Jetzt werden negative Zahlen untersucht, d.h. $a_{16}=1$

$$A^2 = -2^{33} + 2^{32} + 2^{17} + \sum_{i=0}^{15} \overline{1 \cdot a_i} \cdot 2^{i+17} + \sum_{i=0}^{15}\sum_{j=0}^{15} a_i \cdot a_j \cdot 2^{i+j}$$

$$= -2^{33} + 2^{32} + 2^{17} + \left\{\overline{a_{15}} \cdot 2^{32} + \overline{a_{14}} \cdot 2^{31} ... + \overline{a_0} \cdot 2^{17}\right\} + \sum_{i=0}^{15}\sum_{j=0}^{15} a_i \cdot a_j \cdot 2^{i+j}$$

Man sieht, daß die Summe in der geschweiften Klammer maximal den Wert $\{2^{33}\text{-}2^{17}\}$ und minimal den Wert Null annehmen kann, so daß das maximale Ergebnis

$$A^2 = -2^{33} + 2^{32} + 2^{17} + 2^{33} - 2^{17} + \sum_{i=0}^{15}\sum_{j=0}^{15} a_i \cdot a_j \cdot 2^{i+j} = 2^{32} + \sum_{i=0}^{15}\sum_{j=0}^{15} a_i \cdot a_j \cdot 2^{i+j}$$

sein wird. Dafür muß natürlich gelten $a_i=0$ für $i=0..15$, so daß einfach $A^2 = 2^{32}$ ist.

Wenn der Inhalt der geschweiften Klammer Null ist, hat man

$$A^2 = -2^{33} + 2^{32} + 2^{17} + \sum_{i=0}^{15}\sum_{j=0}^{15} a_i \cdot a_j \cdot 2^{i+j} = -2^{33} + 2^{32} + 2^{17} + \sum_{i=0}^{15}\sum_{j=0}^{15} a_i \cdot a_j \cdot 2^{i+j}.$$

Es darf natürlich kein negatives Ergebnis beim Quadrieren entstehen. Immer wenn die geschweifte Klammer Null ergibt, muß $a_i=1$ sein für $i=0..15$. Dann liefert

$$\sum_{i=0}^{15}\sum_{j=0}^{15} a_i \cdot a_j \cdot 2^{i+j} = \left(2^{16} - 1\right)^2 = 2^{32} - 2^{17} + 1 \Rightarrow A^2 = -2^{33} + 2^{32} + 2^{17} + 2^{32} - 2^{17} + 1, \text{ also } A^2 = +1$$

Zwei extreme Zahlenbeispiele sollen den Sachverhalt abschließend illustrieren:

Beispiel 1: $A=-1$

$$a_i = 1 \text{ für } i = 0...15, \Rightarrow A^2 = \left(-2^{16} + 2^{16} - 1\right)^2 = 1$$

$$A^2 = -2^{33} + 2^{32} + 2^{17} + \{0\} + \left\{2^{32} - 2^{17} + 1\right\} = 1$$

Beispiel 2: $A=-2^{16}$

$$a_i = 0 \text{ für } i = 0...15, \Rightarrow A^2 = \left(-2^{16}\right)^2 = 2^{32}$$

$$A^2 = -2^{33} + 2^{32} + 2^{17} + \left\{2^{33} - 2^{17}\right\} + \{0\} = 2^{32}$$

Generell kann ein Quadrierer wesentlich einfacher in Hardware realisiert werden als ein Multiplizierer gleicher Wortlänge.

1.1.4 Der digitale Signalprozessor (DSP) als Grundbaustein der DSV

1.1.4.1 CISC und RISC Konzepte

Bis etwa 1986 war die Fachwelt von einer kaum noch zu übertreffenden Leistungsfähigkeit konventioneller Mikroprozessoren wie dem 68020 oder dem 80386 überzeugt. Diese Prozessoren basieren auf der Philosophie komplexer Befehlssätze (**CISC** ≡ **C**omplex **I**nstruction **S**et **C**omputer), deren Abarbeitung relativ große integrierte Mikroprogramme verlangt. Dafür sollte das **CISC–**Konzept dem Anwender durch entsprechend „mächtige“ Befehle die Programmierarbeit erheblich erleichtern.

Nach und nach kam aber ein ganz anderes Konzept für optimale Prozessorarchitekturen auf, das auf wesentlich vereinfachte Aufbauten mit weniger – aber dafür sehr effizient genutzten - Transistoren abzielte. Es sollten Leistungssteigerungen bis zum Faktor fünf gegenüber herkömmlichen CISC–Maschinen erreicht werden. Die neuartigen Maschinen basierten auf sehr einfachen Befehlssätzen und wurden daher **R**educed **I**nstruction **S**et **C**omputer (**RISC**) genannt.

Umfangreiche Analysen verschiedenartiger Computerprogramme hatten im Vorfeld gezeigt, daß rund 80% der Befehle nur 20% des verfügbaren Befehlssatzes einer typischen CISC–Maschine nutzten. Weitere Untersuchungen führten zu der Erkenntnis, daß die am häufigsten angewandten Instruktionen meist einfache Operationen ausführten und auch die einfacheren Adressierungsarten benutzten. Würde man ausschließlich solche Befehle in besonderen Hardware-Architekturen beschleunigen, könnte man mit einer erheblichen Steigerung der Leistungsfähigkeit rechnen. Die Untersuchungen hatten klar bewiesen, daß sich der bisherige Aufwand zur Implementierung komplexer Befehlen nicht gelohnt hatte. Das Ziel, die Belastung eines Programmierers oder den Compileraufwand zu verringern, war nicht erreicht worden.

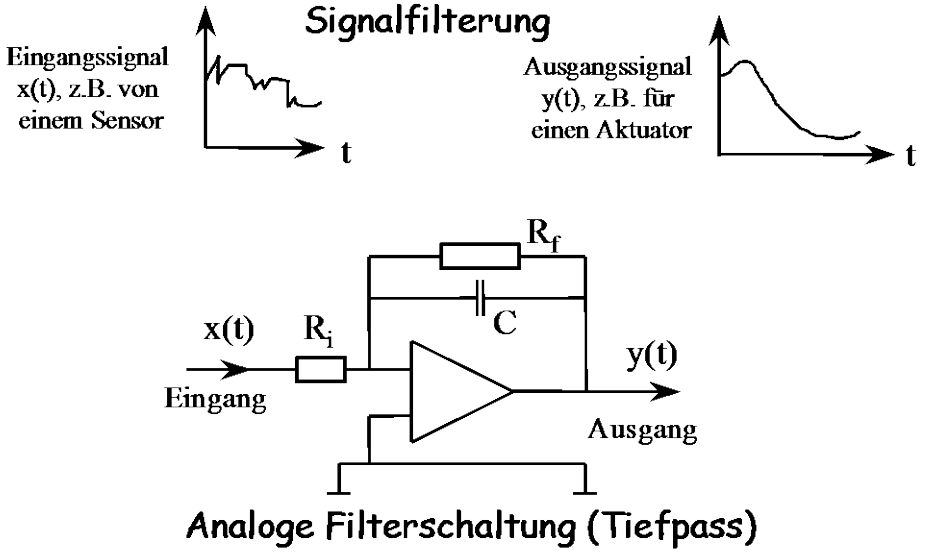
Wenn ein Prozessor nur einfache Befehle hat, kann die Komplexität der zugehörigen Hardware reduziert werden. Daraus folgt, daß mit weniger Transistoren sogar höherer Leistung möglich wird, weil die Arbeitsgeschwindigkeit gesteigert werden kann. Der einfache Befehlssatz gestattet die Abarbeitung eines Befehls in einem einzigen Taktschritt. Komplexe Operationen müssen zwar aus einer Sequenz einfacher Befehle konstruiert werden, wobei aber gegenüber einer fest konfigurierten CISC–Maschine ein weites Feld für Optimierungen verbleibt.

Was kennzeichnet ein RISC-Konzept

- **Alle Befehle laufen in *einem einzigen* Taktschritt ab.**
  Der OP-Code hat stets eine feste Länge, die kleiner oder höchstens gleich der Breite der Datenbusse ist. Zusätzliche Operanden dürfen nicht vorkommen. Die Befehlsdecodierung muß direkt und einfach sein.
- **Speicherzugriffe erfolgen nur bei Lade- und Abspeicherbefehlen**.
  Wenn ein Befehl Speichermanipulationen durchführen muß, sind zwangsläufig mehrere Arbeitsschritte nötig. Ein CISC–Prozessor lädt dazu die Speicherdaten in ein Register, das Register wird manipuliert, und sein Inhalt wird anschließend wieder in den Speicher geschrieben. Eine solche Sequenz benötigt mindestens drei Taktschritte. Ein RISC-Prozessor hingegen bearbeitet Daten typischerweise ***direkt in Registern*** und beläßt sie dort. Daraus folgt, daß beim RISC-Konzept immer eine große Anzahl von Registern benötigt wird.
- Alle Einheiten zur Befehlsausführung sind **festverdrahtet**, d.h. es wird ***kein Mikroprogramm*** implementiert. Es ist praktisch unmöglich, Mikroprogrammierung einzusetzen, wenn man eine Befehlsausführung in einem Taktschritt zu realisieren hat.

1.1.4.2 Digitale Signalverarbeitung mit Echtzeitanforderungen

Ohne tieferen Einblick denkt man beim Begriff ***digitale Signalverarbeitung*** zunächst wohl an die Abarbeitung komplizierter Algorithmen, wie sie schon bei recht einfachen Aufgaben wie der Filterung von Signalen vorkommen. Der DSV haftet der Ruf abschreckender mathematischer Komplexität an. Eine Ursache dafür ist, daß z.B. der nachrichtentechnisch orientierte Elektroingenieur in der Vergangenheit im Rahmen seiner Ausbildung und Berufspraxis z.B. mit analogen Filtern, Modulatoren und Demodulatoren und deren recht einfacher mathematischer Beschreibung vertraut war. So ist z.B. ein analoges Tiefpaßfilter, bestehend aus einem Operationsverstärker, zwei Widerständen und einem Kondensator anhand einer einfachen Formel in seiner Funktion zu verstehen. Der Entwurf ist leicht und die Realisierung sehr kostengünstig – wie Bild 1.9 zeigt.



Mathematischer Zusammenhang für die analoge Signalverarbeitung

Ansatz mit komplexen Amplituden:

$$\frac{\underline{Y}}{\underline{X}} = -\frac{R_f}{R_i}\cdot\left[\frac{1}{1+j\omega R_f\cdot C}\right],\ \text{mit}\ y(t)=\mathrm{Re}\left\{\underline{Y}\cdot e^{j\omega t}\right\}, x(t)=\mathrm{Re}\left\{\underline{X}\cdot e^{j\omega t}\right\}$$

Gesamtaufbau bei digitaler Signalverarbeitung

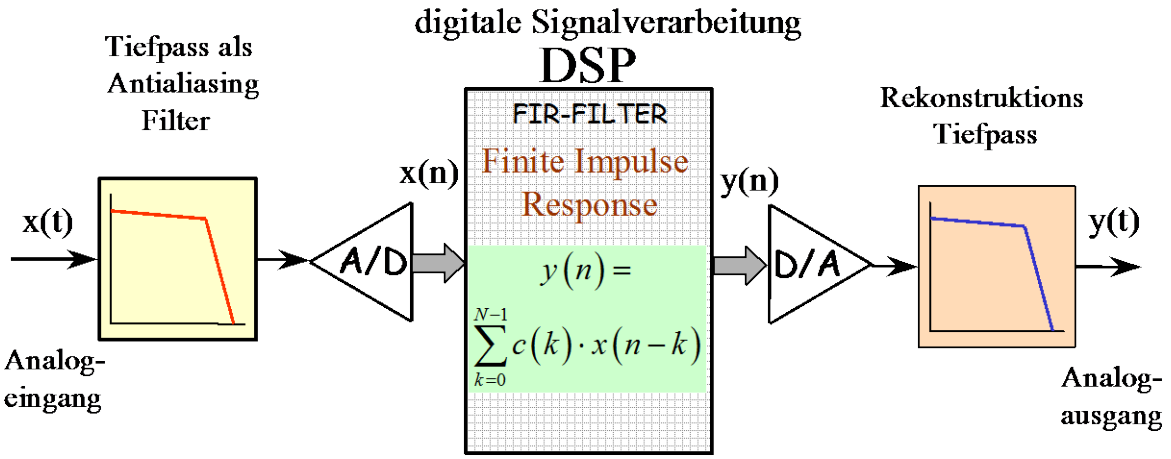


Bild 1.9: Gegenüberstellung von analoger und digitaler Signalverarbeitung

Das digitale Äquivalent hingegen benötigt mehrere elektronische Stufen zur Wandlung des analogen Eingangssignals in digitale Eingangsdaten, zur Verarbeitung der Daten und zur Rekonstruktion eines in der Regel benötigten analogen Ausgangssignals. Die Datenverarbeitung umfaßt meist eine Vielzahl digitaler Multiplikationen und Addition (MAC–Operationen), die nur mit umfangreicher Hardware in der benötigten Geschwindigkeit verfügbar sind.

Wo liegen unter diesen Aspekten die Vorteile digitaler Signalverarbeitung?

- Die digitale Signalverarbeitung (DSV) leidet nicht unter Drift- und Alterungseffekten der Komponenten. Parameterstreuungen, die bei der Serienherstellung analoger Bauteile unvermeidlich sind, müssen in Analogschaltungen durch oft aufwendige Abgleichmaßnahmen unschädlich gemacht werden. Bei Digitalschaltungen zeigt bereits der Chiptest, der in der Halbleiterfertigung noch vor dem Gehäuseeinbau durchgeführt wird, ob der Baustein alle Anforderungen auch hinsichtlich der vorgesehenen Arbeitsgeschwindigkeit (Taktfrequenz) fehlerfrei erfüllt. Es gibt nichts abzugleichen und weder Drift noch Alterung spielen im Rahmen der üblichen Bauteillebensdauer unter „Normalbedingungen“ eine Rolle.
- Digitalbausteine haben von Natur aus eine hohe ***Störresistenz,*** denn alle rauschähnlichen Störungen, die betragsmäßig unterhalb der Schaltschwelle (bei CMOS z.B. 2,5V) bleiben, werden komplett unterdrückt. Auch nicht perfekt „gesiebte“ Versorgungsspannungen verursachen sind in Digitalschaltungen keine Probleme.
- Des weiteren kann in einem DSP auf einfache Weise die Möglichkeit eines Selbsttests implementiert werden. So kann ohne Ausbau aus der Schaltung in regelmäßigen Abständen die Funktionstüchtigkeit überprüft werden. Das ist nicht nur in sicherheitsrelevanten Anwendungen ein unschätzbarer Pluspunkt. Zudem sind z.B. bei Filteranwendungen die Filterkoeffizienten und damit die Filtercharakteristiken jederzeit änderbar. Es kann sogar zwischen völlig verschiedenen Funktionen per Software „umgeschaltet“ werden. Ein digitales Filter kann damit adaptiv auf Änderungen von Signalparametern wie z.B. Amplitude oder spektrale Zusammensetzung reagieren.

Die Schlüsseloperation der DSV wird durch die Abkürzung **MAC** (**M**ultiply and **Ac**cumulate) charakterisiert. **MAC–**Operationen sind Kernstücke der Faltung, der Korrelation sowie der Fourier-, Laplace- und auch der Wavelet–Transformation.

Die Korrelation wurde anhand von Bild 1.8 und der Gleichung (1.1) eingeführt. Die Rechenleistung, die für eine derartige einfache Operation erforderlich ist, kann enorm sein, wenn hohe Arbeitsgeschwindigkeit für ***Echtzeitbetrieb*** gefordert wird. In einem Kommunikationssystem muß der Empfänger das ankommende Empfangssignal lückenlos verarbeiten können. Im Fall des Korrelators muß die MAC–Geschwindigkeit somit der Abtastrate des A/D-Wandlers entsprechen.

Weitaus höhere Anforderungen an die MAC–Geschwindigkeit stellen digitale Filter, insbesondere FIR–Strukturen. Die Grenzen der Leistungsfähigkeit eines DSP lassen sich deshalb sehr gut an einem FIR–Filter aufzeigen. Die Eigenschaften eines solchen Filters nach Bild 1.10 werden durch die Zahl *N* der Anzapfungen (Taps) und die Filterkoeffizienten *c(i)* bestimmt. In einem *N*-stufigen digitalen Filter werden *N* Abtastwerte des Eingangssignals *x(n)* gespeichert. Die Verzögerungen sind in einfacher Weise mit Flip-Flops zu realisieren, deren Taktfrequenz durch die Abtastrate vorgegeben ist. Zur Berechnung ***eines*** Ausgangswertes *y(n)* müssen mit jedem Abtasttakt ***alle gespeicherten*** Eingangswerte mit den zugehörigen Filterkoeffizienten *c(i)* multipliziert und alle Ergebnisse der Multiplikationen aufsummiert werden. Formelmäßig bedeutet dies

$$y(n) = \sum_{k=0}^{N-1} c(k) \cdot x(n-k). \qquad (1.13)$$

Obwohl (1.13) der Gleichung (1.1) sehr ähnlich sieht, gibt es einen beträchtlichen Unterschied. Denn während im Fall des Korrelators die MAC–Geschwindigkeit lediglich der Abtastrate des A/D-Wandlers entsprechen muß, erfordert das Auftreten eines jeden neuen Abtastwertes *x(n)* aus dem A/D-Wandler in Bild 1.10 die komplette Berechnung der Summe nach (1.13), d.h. *N* Multiplikationen und *N* Akkumulationen.

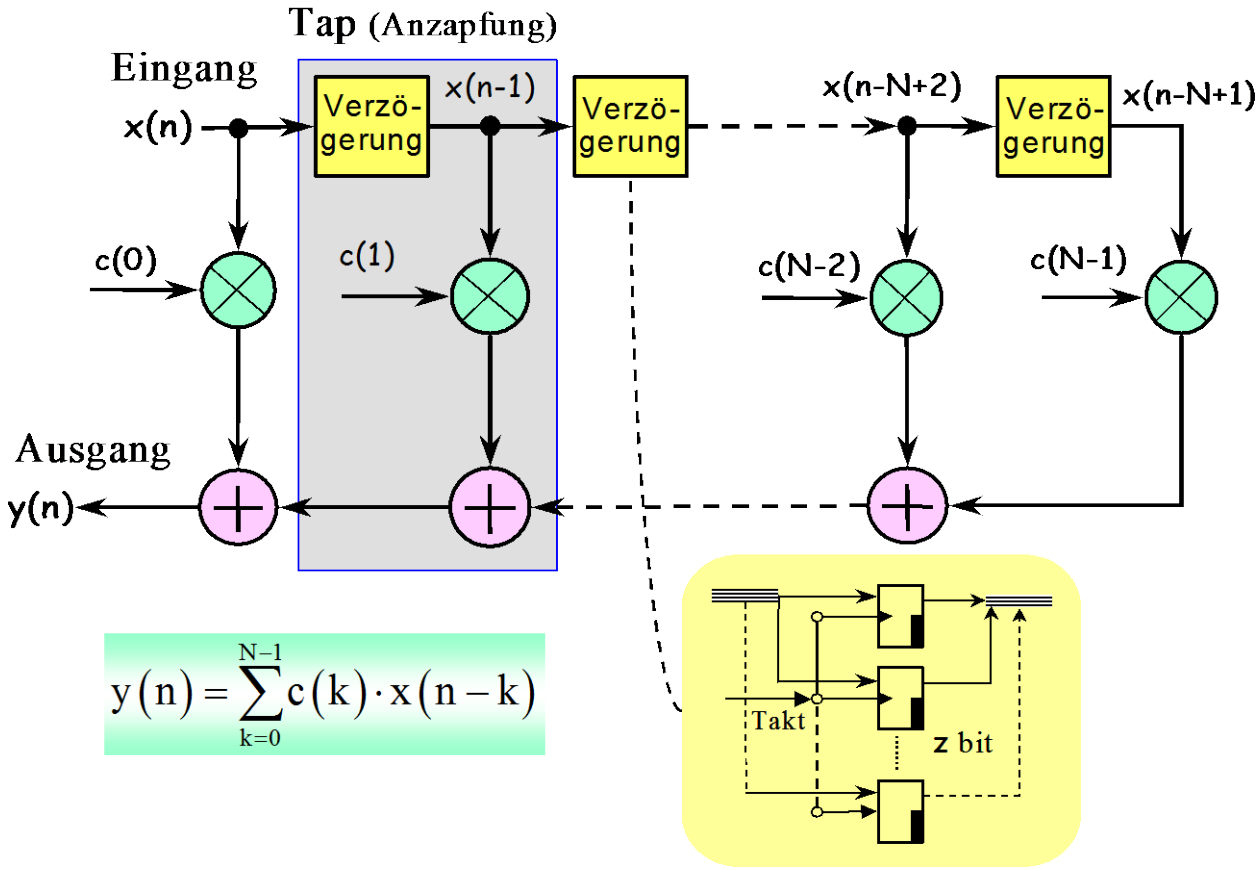


Bild 1.10: Struktur eines digitalen FIR–Filters (Finite Impulse Response)

Ein typisches Kennzeichen digitaler Filter ist eine hohe Anzahl von Taps, z.B. 100 und mehr. Da ein gewöhnlicher DSP aber nur eine MAC–Operation pro Maschinenzyklus (≡ Taktzyklus) ausführt, würde bei einer Abtastrate von 1MHz (Grenzfrequenz des zu verarbeitenden Signals $f_g <$ 500kHz) für N=100 bereits eine Rechenleistung von $100 \cdot 10^6$ Instruktionen pro Sekunde, also 100 MIPS benötigt. Moderne DSPs bieten – bei realistischer Betrachtungsweise – maximal etwa 1000 MIPS.

Obwohl man DSV prinzipiell mit ***jedem Mikrorechner*** durchführen könnte, wären die Resultate meistens enttäuschend. Wegen der typischen Architektur von Standard-Mikroprozessoren oder Mikrocontrollern ließen sich höchstens Signalfrequenzen von nur ***wenigen hundert Hertz*** bewältigen. Für eine leistungsfähige DSP–Architektur werden daher, grundlegend neue Ansätze benötigt. Dabei werden andererseits aber bewährte Grundstrukturen nicht verworfen, sondern weiterentwickelt und an die neuen Aufgaben angepaßt. Die wesentlichen Charakteristika, die einen DSP ausmachen und ihn vom Mikroprozessor oder Mikrocontroller unterscheiden, werden im folgenden am Beispiel der DSP5600x-Famile des Herstellers Motorola betrachtet.

Der DSP56000 hat eine Datenwortlänge von 24 bit und verfügt über eine **Harvard**-Architektur mit **vier** separaten internen Bussen für Daten- und OP-Code-Transfer (GDB ≡ General Purpose, z.B. für Bitmanipulationen; PDB ≡ Programm; XDB ≡ X-Datenspeicher; YDB ≡ Y-Datenspeicher). Drei separate interne **16bit-Adressbusse** dienen dem parallelen Zugriff auf Programmspeicher (PAB), X-Datenspeicher (XAB) und Y-Datenspeicher (YAB). Die Speicherorga-

nisation gewöhnlicher Mikroprozessoren und Mikrocontroller ist zum Vergleich in Bild 1.11 dargestellt. Im einfachsten Fall hat man es mit der so genannten „von Neumann"–Architektur zu tun (links im Bild), wobei nur ***ein einziger Speicher*** mit zugehörigen Daten- und Adreßbus vorhanden ist. Im Speicher werden sowohl das Programm als auch Daten abgelegt. Der Programmierer muß für eine geeignete Aufteilung selbst sorgen und geeignete ***Schutzmechanismen*** gegen ungewolltes Überschreiben von Programmteilen vorsehen. Der schwerwiegendste Nachteil dieser Architektur offenbart sich beim Datentransfer. Da nur ein Bussystem vorhanden ist, erfordert jeder Datentransfer mehrere Schritte, d.h. auch mehrere Maschinenzyklen, weil sowohl der OP-Code eines entsprechenden Befehls als auch die zu transportierenden Daten über ***dasselbe Bussystem*** übertragen werden müssen. Dies kann natürlich hier nur sequentiell geschehen. Auch die rechts in Bild 1.11 dargestellte Architektur mit zwei Speichern, die per Hardware bereits in Programm- und Datenspeicher aufgeteilt sind, hebt den entscheidenden Nachteil nicht auf, da es auch hier nur ein Bussystem gibt, das sequentiell benutzt werden muß.
Einen erheblichen Vorteil bringt dagegen eine ***Harvard-Architektur*** nach Bild 1.12, mit deren Hilfe drei parallele Transfers von Information möglich sind. Typischerweise dient ein Bussystem dem OP-Code-Transfer, während die beiden andern Daten transportieren. Genau diese Konstellation wird bei den meisten Algorithmen der DSV, die MAC–Operationen beinhalten, benötigt. Denn im allgemeinen Fall müssen für jede auszuführende Multiplikation zwei neue Operanden zur Verfügung stehen. Die in Bild 1.12 dargestellt Harvard-Architektur erlaubt somit z.B. die Programmierung einer ***Korrelation*** nach (1.1) mittels einer Schleife, die nur ***einen einzigen Befehl*** enthält und somit in einem Maschinenzyklus abläuft. Die Instruktion

```
MAC X0,Y0,A    X:(R0)+,X0    Y:(R4)+,Y0
```

kennzeichnet wohl am Besten die typischen Merkmale eines DSP, die ihn vom Standard-Mikroprozessor unterscheiden.

- die MAC–Operation
- die Harvard-Architektur

Ein weiteres wichtiges Merkmal, das im weiteren behandelt wird und für ein Vielzahl von DSV-Algorithmen – also auch für (1.1) – benötigt wird, ist das „Zero Overhead Looping", d.h. eine Ausführung von Schleifen, ohne daß dafür zusätzliche Maschinenzyklen anfallen. Man spricht auch von Hardware-DO-Loops, weil alle benötigten Funktionen in Hardware implementiert sind. Die Beziehung (1.1) kann damit in zwei Zeilen programmiert werden – ein Beispiel folgt weiter unten.
Die Adreßbusse des DSP nach Bild 1.13 werden aus der AGU ≡ **A**ddress **G**eneration **U**nit gespeist, deren Blockschaltung in Bild 1.14 dargestellt ist, während Bild 1.15 das Programmiermodell zeigt. In der AGU gibt es ***drei Registergruppen***, die aus jeweils ***8 Registern*** aufgebaut sind.

- Adreßzeiger-Register R0…R7,
- Adreß-Offsetregister N0…N7,
- Adreß-Modifikationsregister M0…M7.

Die AGU gestattet verschiedene Adressierungsarten für den Datenzugriff. Funktionell ist sie ***zweigeteilt***, um gleichzeitig Datenzugriffe im X- und Y-Datenspeicherbereich zu ermöglichen. Bei der Adreßberechnung werden normale ***Zweierkomplement***-Arithmetik, ***Modulo***-Arithmetik und ***'Reverse-Carry'***-Arithmetik unterschieden. Letztere findet ausschließlich bei der Ausführung von Algorithmen der FFT (Fast Fourier Transform) Anwendung. Da gemäß Bild 1.14 zwei ALUs mit je 3 Volladdierern verfügbar sind, können in einem Maschinenzyklus zwei Adreßberechnungen stattfinden, und zwar für jede 2-er-Kombination aus den Adreßbussen XAB, YAB, PAB.

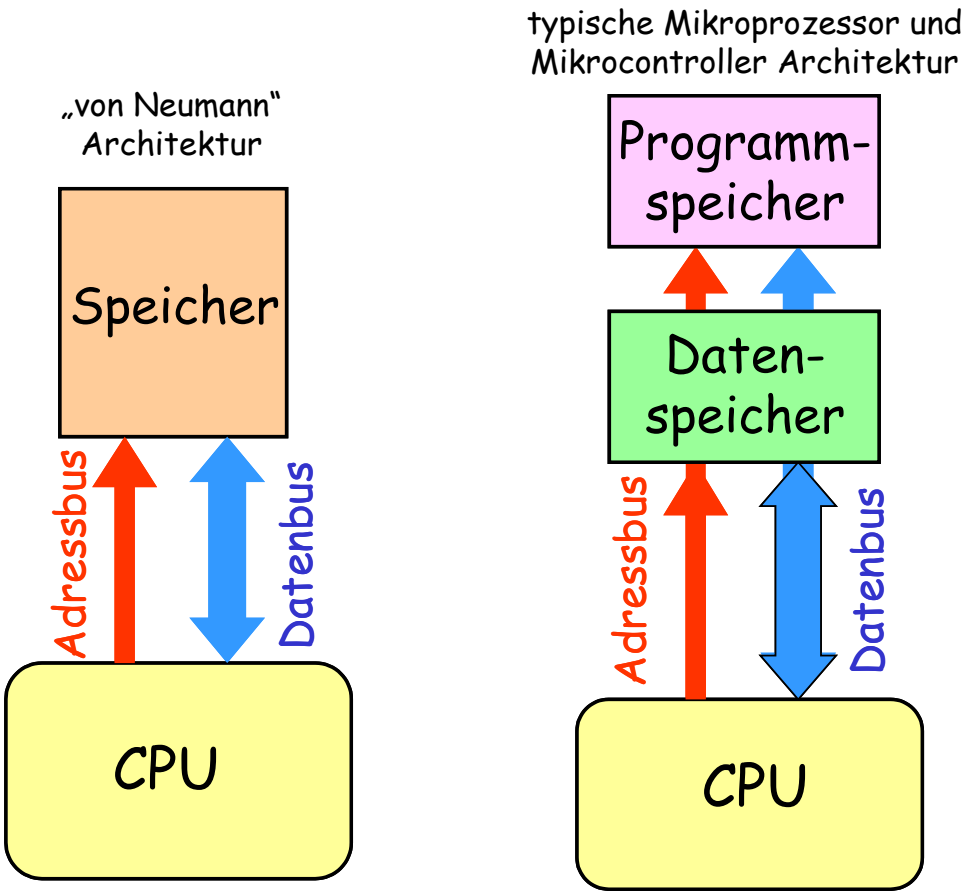


Bild 1.11: Speicherorganisation gewöhnlicher Mikroprozessoren und Mikrocontroller

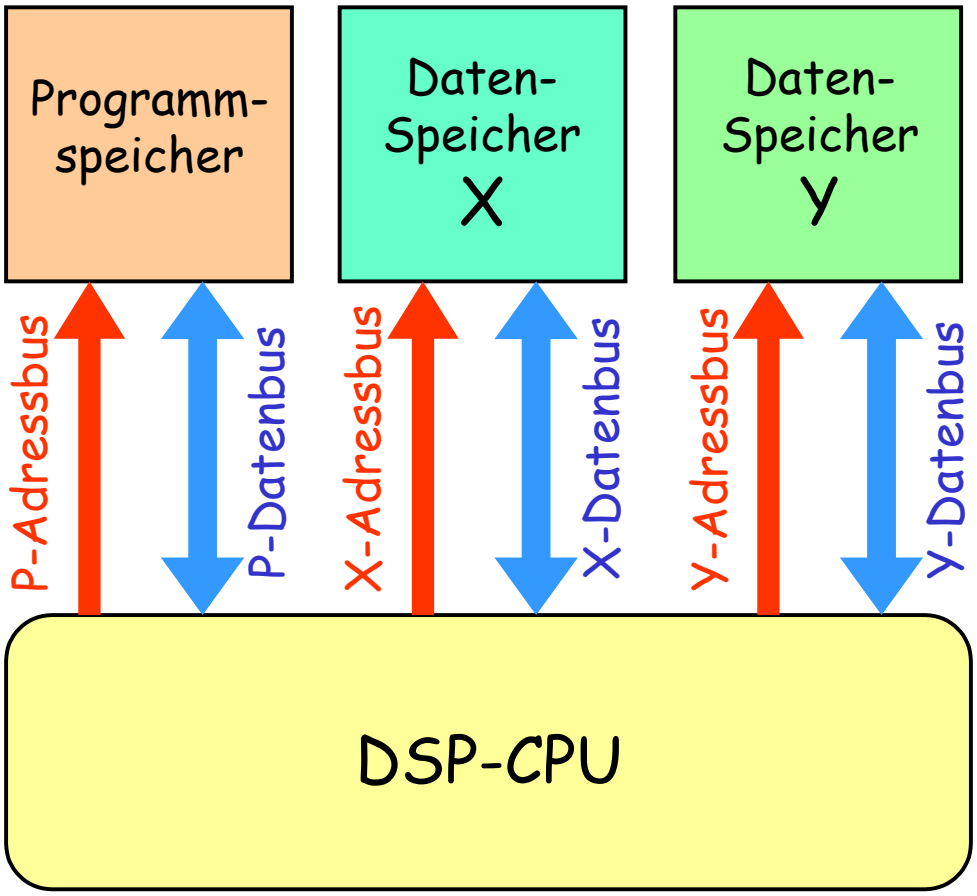


Bild 1.12: Typische Harvard-Architektur eines DSP

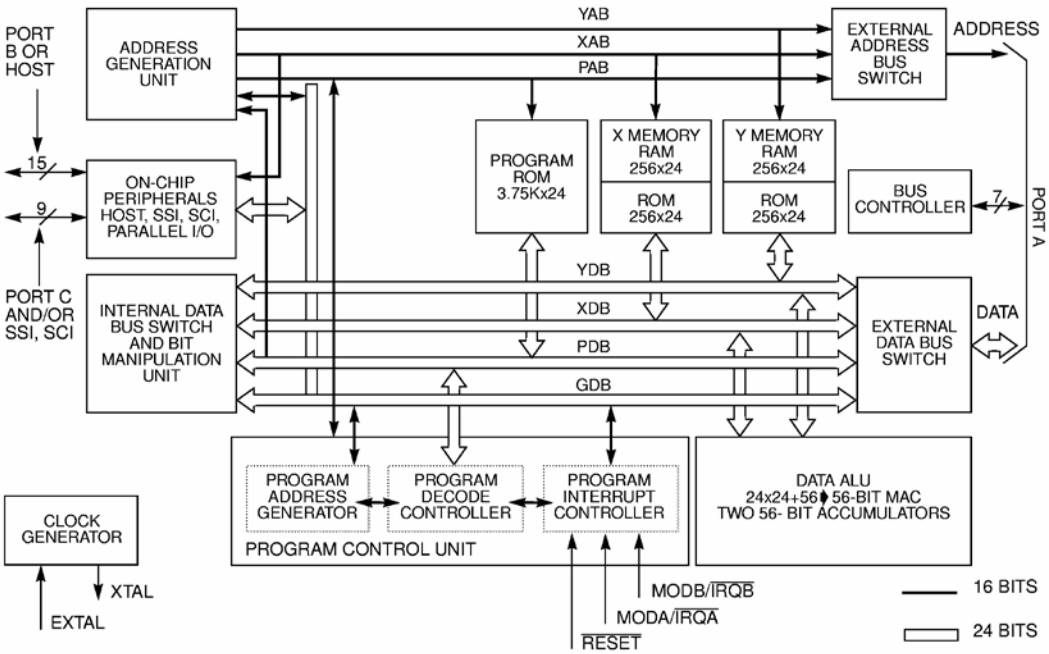


Bild 1.13: Blockdiagramm des DSP 56000

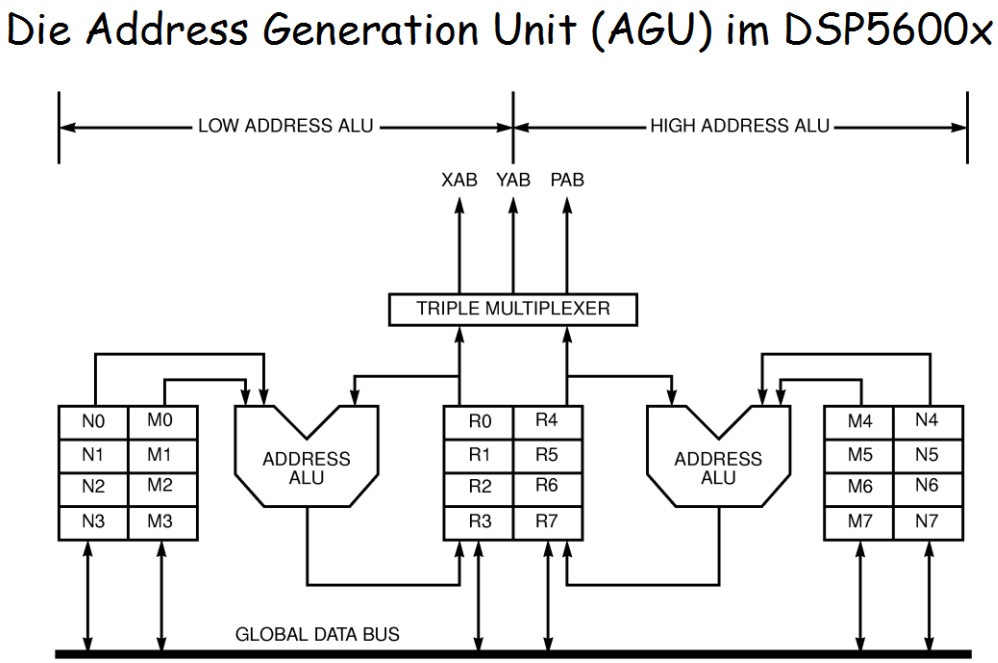


Bild 1.14: Aufbau der Adreßerzeugungseinheit im DSP 5600x

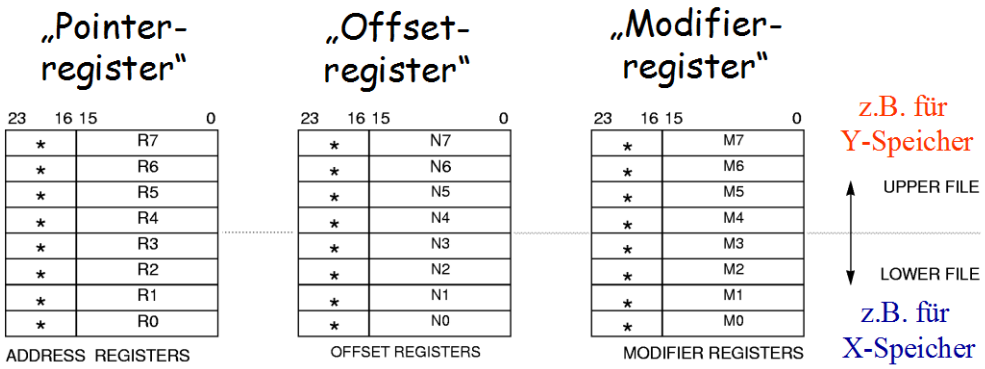


Bild 1.15: Das Programmiermodell der AGU

Es gibt folgende 3 Adressierungsarten

- Adreßregister direkt
- Adreßregister indirekt
- spezial

Beispiele für indirekte Registeradressierung

- (R_n) indirekt ohne Pointermanipulation
- $(R_n)+$ Postinkrement
- $(R_n)-$ Postdekrement
- $-(R_n)$ Prädekrement
- $(R_n)+N_n$ Postinkrement mit Offset
- $(R_n)-N_n$ Postdekrement mit Offset
- (R_n+N_n) 'indexed' durch Offset: EA[3] <= Inhalt von R_n+N_n

Anwendungsbeispiele

```
MOVE  A1,X:(R0)
```

bringe A1 an den Platz im X-Datenspeicher, der von R0 adressiert wird

R0 enthält X-Adresse, z.B. $1000
N0 beliebig
M0 $FFFF

[3] effektive Adresse

```
MOVE    B0, Y:(R1)+
```

bringe B0 an den Platz im Y-Datenspeicher, der von R1 adressiert wird und erhöhe anschließend R1 um Eins ***(Postinkrement)***

entsprechend:

```
MOVE    B0,Y:(R1)-
MOVE    X1,X:(R2)+N2
```

Bringe zuerst X1 an den X-Speicherplatz, der von R2 adressiert wird. Dann wird R2 um den Inhalt von N2 erhöht und beim nächsten Aufruf der Instruktion kommt X1 an den X-Speicherplatz, der vom neuen Inhalt von R2:=$R2_{alt}$+N2 adressiert wird.

Modulofunktionen zur Konstruktion von Ringspeichern

Der ***M***-Registersatz bietet besondere Funktionen zum effizienten Aufbau von Schleifen in einem Datenspeicher. Hier wird die Modulofunktion betrachtet.

Register M_n	**Funktion**
0000	Reverse-Carry ⇒ für FFT
0001	Modulo 2
0002	Modulo 3
m	Modulo (m+1)
.....	.....
32.767	Modulo 32.768
.....	reserviert
65.535 $FFFF	Modulo 65.536 (linear) ⇒ nach Reset

Wenn ein Modifierregister $\boldsymbol{M_n}$ den Inhalt m hat, dann ist der 'Modulus' ***M=m*+1**.

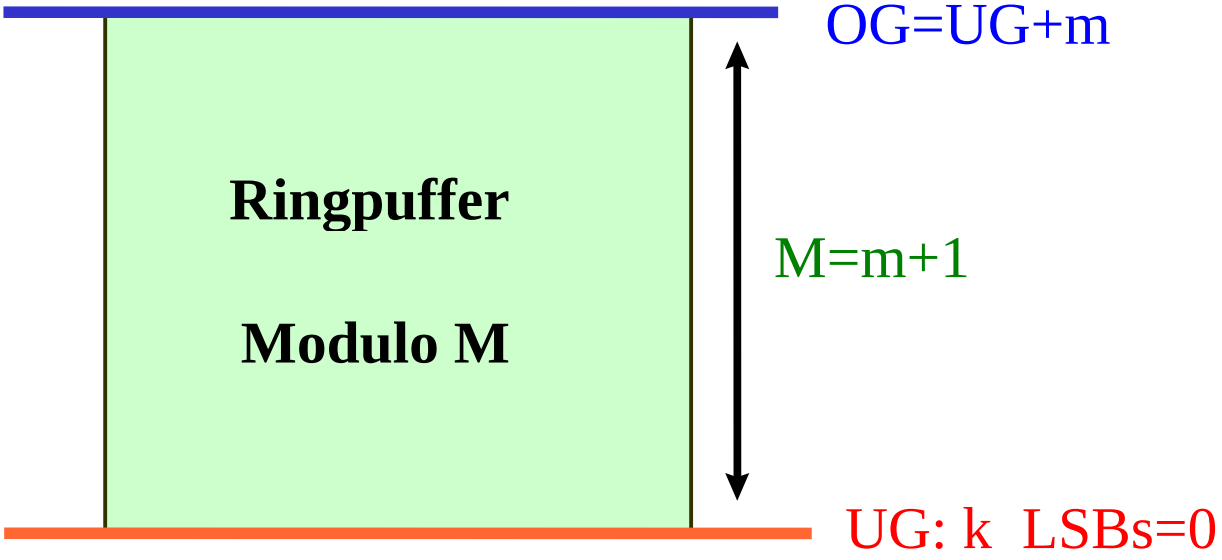


Bild 1.16: Implementierung eines Ringpuffers mit Hilfe der **M**-Register der AGU

Die Untergrenze **UG** ist nicht identisch mit den Pointer $\mathbf{R_n}$, sondern erfordert Nullen in den unteren ***k*** Bits, wobei $2^k \geq M$ sein muß.

Beispiel: **M=21∈(16...32) ⇒ k=5 =>UG=xxxx00000**

Der Ringpuffer kann mit $\mathbf{(R_n) \pm 1}$ durchlaufen werden, oder auch mit größerer Schrittweite, z.B. mit einem Offset $\pm N_n$.

Beispiel

MOVE	**X0, X:(R2)+N2**
M2=20	Modulus: **M=21**
N2=10	Offset
R2=75	Pointer

wegen ld{21} > 4 und <5 ist k = 5
Die Untergrenze hat also **mindestens** fünf Nullen in den LSBs.
Der Pointer **R2** liegt im Bereich 2^6=64....2^7=128
Die Modulo-Buffer-Untergrenze ist somit **64**.

Untergrenze UG: 64 → Obergrenze OG: 64+20 = 84

Ablauf:	**X0 → X:(75)**
(R_n)+N2	**X0 → X:(85) >84 geht nicht, da 85 außerhalb des Buffers ist**
Aktion:	**Modulus wird abgezogen**
$R2_{neu}$:	**R2+N2-(M2+1) => 75+10-(21)=64 (Untergrenze)**
weiter:	**R2: 64+10 = 74**
	R2: 74+10 = 84 (Obergrenze)
$R2_{neu}$:	**R2+N2-(M2+1) => 84+10-(21)=73**

1.1.4.3 Bitreversing (Reverse–Carry–Arithmetik) für die schnelle Fouriertransformation

Diskrete Fouriertransformation (DFT)

$$G\left(\frac{n}{NT}\right) = \sum_{k=0}^{N-1} g(kT) \cdot e^{-j2\pi nk/N} \quad \text{mit } n = 0,1...N-1 \tag{1.14}$$

Inverse diskrete Fouriertransformation (IDFT)

$$g(kT) = \frac{1}{N} \sum_{n=0}^{N-1} G\left(\frac{n}{NT}\right) \cdot e^{j2\pi nk/N} \quad \text{mit } k = 0,1...N-1 \tag{1.15}$$

Formaler Ansatz für die schnelle Fouriertransformation (FFT)

$$X(n) = \sum_{k=0}^{N-1} x_o(k) \cdot e^{-j2\pi nk/N} \quad \text{mit } n = 0,1...N-1$$

mit der Abkürzung: $W = e^{-\frac{j2\pi}{N}}$ wird

$$X(n) = \sum_{k=0}^{N-1} x_o(k) \cdot W^{nk}, \quad \text{mit } n = 0,1...N-1 \qquad (1.16)$$

Ausgangsvektor Eingangsvektor

$$W^{nk} = e^{-j2\pi\frac{nk}{N}} = \cos(2\pi nk/N) - j\sin(2\pi nk/N), \text{ mit } \quad k,n \in \{0,1,...N-1\}$$

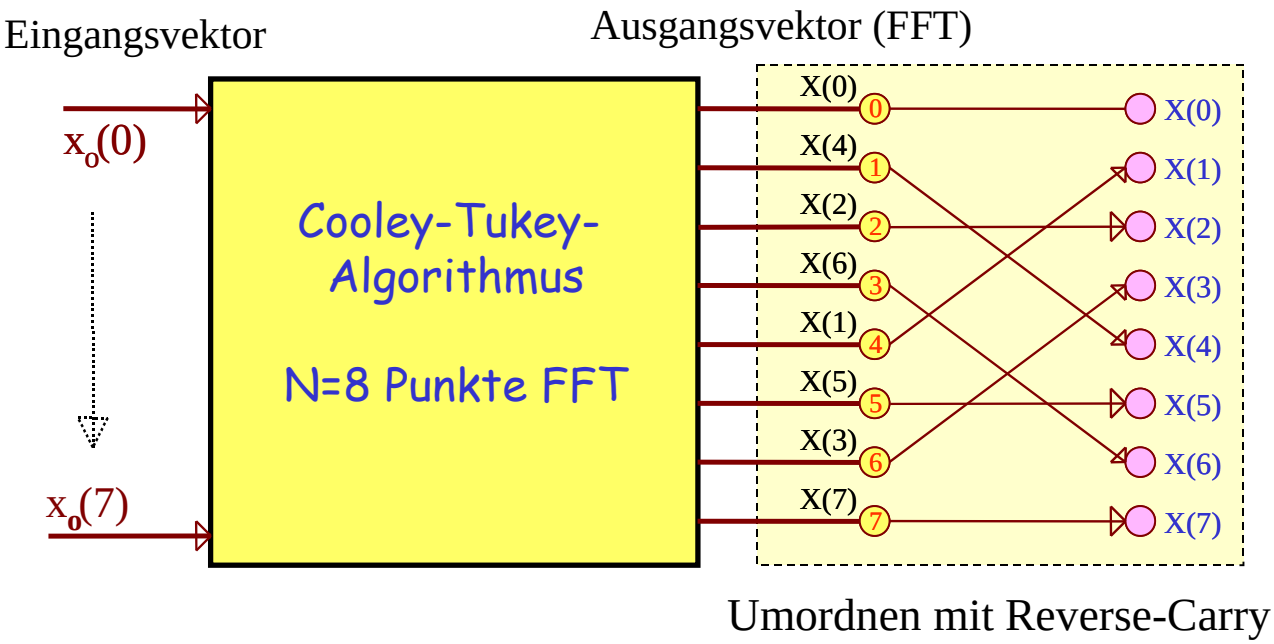
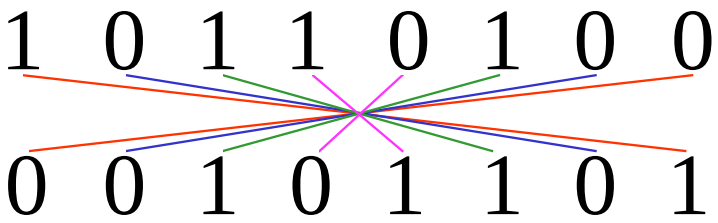


Bild 1.17: Vereinfachter Signalflußgraph für eine 8-Punkte FFT

Programmierung der Reverse-Carry-Funktion

```
Mn=$0000
```

Wirkung des ’Reverse-Carry’ auf die Bitanordnung:



Beispiel zur Reverse-Carry-Arithmetik

Das Register M_n wird auf \$0000 gesetzt.

- R_n - Inhalt wird „reversiert“, d.h. das MSB wird zum LSB und entsprechend wird Bit für Bit im Platz vertauscht
- N_n - Inhalt wird ebenfalls „reversiert“; bei N_n:=$2^{(k-1)}$ hat man dann immer den Wert ...01
- es wird normal binär addiert
- das Ergebnis wird „reversiert“ $\Rightarrow$ jetzt hat man den endgültigen Adreßzeiger

Es wird eine Basis-2-FFT betrachtet, d.h. das Offset-Register N_n enthält eine Zweierpotenz.

hier: N=8 $\Rightarrow$ k=3; N_n hat den Inhalt $2^{(k-1)}$ = 100

R_n enthalte den Startwert 000

2^2	2^1	2^0	Operation	Adreßpointer
0	0	0	R_n	X(0) $\Leftrightarrow$ X(0)
0	0	1	N_n reversiert	
0	0	1	$R_n + N_n$	
1	0	0	Summe reversiert	X(4) $\Leftrightarrow$ X(1)
0	0	1	R_n	
0	0	1	N_n reversiert	
0	1	0	$R_n + N_n$	
0	1	0	Summe reversiert	X(2) $\Leftrightarrow$ X(2)
0	1	0	R_n	
0	0	1	N_n reversiert	
0	1	1	$R_n + N_n$	
1	1	0	Summe reversiert	X(6) $\Leftrightarrow$ X(3)
0	1	1	R_n	
0	0	1	N_n reversiert	
1	0	0	$R_n + N_n$	
0	0	1	Summe reversiert	X(1) $\Leftrightarrow$ X(4)
1	0	0	R_n	
0	0	1	N_n reversiert	
1	0	1	$R_n + N_n$	
1	0	1	Summe reversiert	X(5) $\Leftrightarrow$ X(5)
1	0	1	R_n	
0	0	1	N_n reversiert	
1	1	0	$R_n + N_n$	
0	1	1	Summe reversiert	X(3) $\Leftrightarrow$ X(6)
1	1	0	R_n	
0	0	1	N_n reversiert	
1	1	1	$R_n + N_n$	
1	1	1	Summe reversiert	X(7) $\Leftrightarrow$ X(7)

1.1.4.4 Die ALU im DSP 56000

Die entscheidende Rechenleistung eines DSP wird in der Daten–ALU bereitgestellt. Im hier betrachteten Beispiel enthält sie zwei 56-bit-Akkumlatoren ***A*** und ***B***. Beide Akkus sind aus drei Untereinheiten A0, A1 und A2 bzw. B0, B1 und B2 zusammengesetzt. A2 und B2 waren bei den ersten 56000-Bausteinen nur 8 bit lange Erweiterungen.

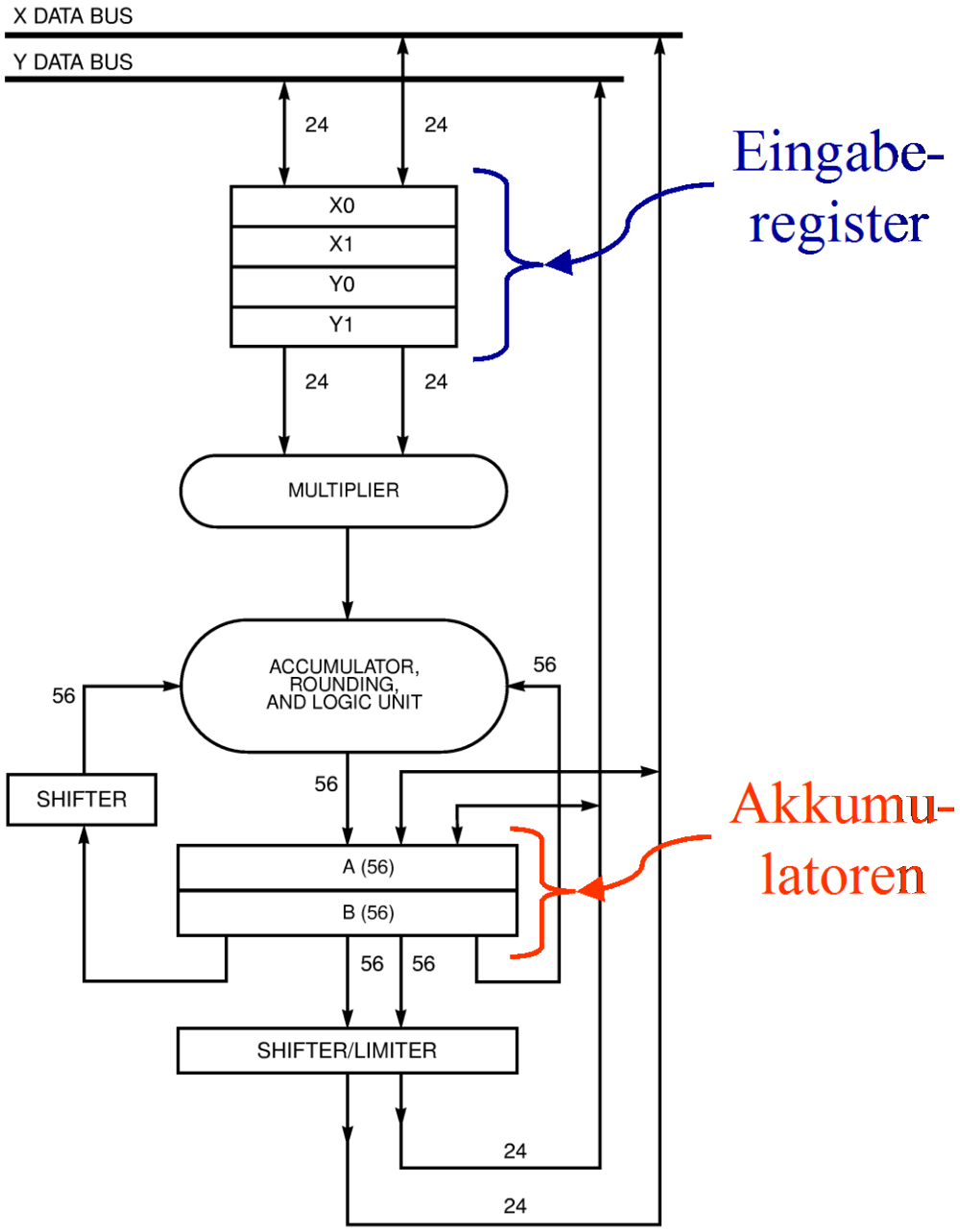


Bild 1.18: Blockdiagramm der ALU–Funktionen

Bei den meisten Nachfolgetypen wurden für A2 und B2 ebenfalls 24 bit implementiert. Bei einem 48-bit-Wort enthalten die Register A1 und B1 jeweils das **MSW** (≡ **M**ost **S**ignificant 24-bit-**W**ord). A0 und B0 enthalten entsprechend das **LSW** (≡ **L**east **S**ignificant 24-bit-**W**ord).

Der DSP verfügt in der ALU des weiteren über vier Register X0, X1 und Y0, Y1 mit jeweils 24bit Länge, die bei Bedarf zu zwei 48 bit Registern X und Y verbunden werden können. Diese Register können Daten über ihre zugehörigen Busse lesen oder ausgeben und enthalten die Quelloperanden von **MAC–**Operationen. Bei der Ausführung eines **MAC–**Befehls werden zwei 24 bit-Werte, z.B. aus den Registern X0 und Y0, miteinander multipliziert, und das 48bit-Ergebnis wird einem der 56 bit Akkus (A oder B) additiv oder subtraktiv zugeführt. Gleichzeitig können die involvierten X und Y-Register durch parallele Datentransfers aus den zugehörigen Datenspeichern neu geladen werden. Dies läuft alles in ***einem einzigen Maschinenzyklus*** ab, d.h. wenn der DSP mit 200MHz getaktet wird, in 5ns.

In der ALU gibt es ferner zwei „Schiebeeinrichtungen“ (Shifter) zur Bitmanipulation und für die dynamische Skalierung von Festkommadaten – siehe Bild 1.18. Zudem sorgt ein Begrenzer (Limiter) dafür, daß grobe arithmetische Fehler, die normalerweise bei binären Überläufen (...11111 ⇒...00000) entstehen würden, vermieden werden. Anstatt des Überlaufs nach ...00000 wird durch den Limiter der Maximalwert, der mit der verfügbaren Wortbreite darstellbar ist, festgehalten. Dieser Zustand wird mit einem „Flag“ im **C**ondition **C**ode **R**egister **(CCR)** signalisiert.

1.1.4.5 Das Programmiermodell für die ALU des DSP56000

Bei der Programmierung eines DSP5600x wird die Verwandtschaft mit der 16-bit-Mikroprozessorarchitektur des 68000 (Motorola) deutlich. Der Registersatz des DSP56000 gliedert sich in drei Hauptbereiche, die jeweils den Funktionseinheiten AGU (Adreßberechnungs-Einheit), ALU und dem Programm-Steuerwerk (Control Unit CU) zugeordnet sind – vgl. auch Bild 1.22.

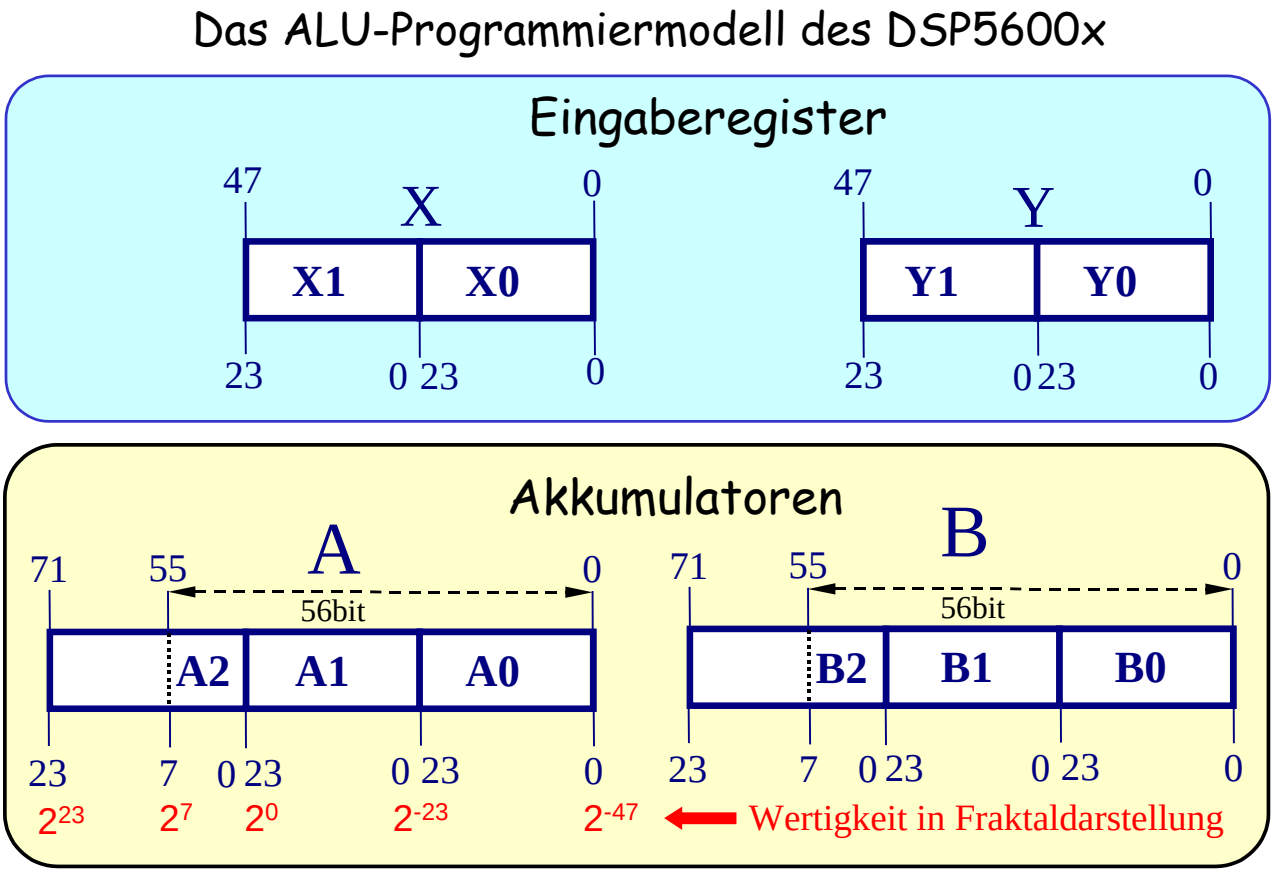


Bild 1.19: Modell zur ALU–Bedienung im DSP 5600x

Die CU wird im nächsten Abschnitt betrachtet.

Die Daten-ALU sieht mit den beiden Akkumulatoren und der X/Y-Registerkombination einer Mikroprozessor-Architektur sehr ähnlich. Diese aufgabenorientierte Struktur resultiert aus der Spezialisierung von DSPs auf Multiplizier-Akkumulier-Operationen, die bekanntlich in der DSV die Schlüsselrolle spielen. Bild 1.19 gibt einen Überblick über Aufbau und Verwendung der Akkumulatoren **A** und **B** sowie der Register **X** und **Y**. Die Register **X** und **Y** liefern die Eingangsdaten für die ALU. Sie können als vier separate 24-bit-Register **X0**, **X1**, **Y0** und **Y1** verwendet werden oder auch als zwei 48bit Register **X** und **Y**, die jeweils durch Aneinanderreihen von **X1X0** und **Y1Y0** definiert werden können. **X1** und **Y1** enthalten jeweils die höherwertigen Datenworte. Die Register dienen als Pufferspeicher zwischen den X- und Y-Datenbussen und der **MAC–**Einheit. Auf diese Weise liefern sie die Quelloperanden an die ALU, während gleichzeitig neue Operanden für den nächsten auszuführenden Befehl geholt werden können. Über die zugehörigen Datenbusse kann der Registerinhalt auch gelesen werden.

Die Akkumulatoren sind vorzugsweise Ziele von **MAC–**Operationen. Generell sind drei Operanden in eine **MAC–**Operation eingebunden: Eingangsseitig je ein 24bit Wort aus den **X**- und **Y**-Registern (**X0,X1,Y0,Y1**). Es wird eine vorzeichenbehaftete Multiplikation in fraktaler Zweierkomplementdarstellung durchgeführt, wobei das 48bit lange Produkt in Fraktaldarstellung zum Inhalt eines der Akkumulatoren **A** oder **B** addiert wird (dritter Operand). Das Ergebnis wird im entsprechenden Akku in einer Wortlänge von 56bit gespeichert – von dieser Wortlänge stammt im übrigen die Bezeichnung „DSP 56000“. Die (ältere) 8-bit-Erweiterung der Akkus erlaubte das Erfassen von maximal 256 Überläufen. Dies erwies sich in vielen Anwendungen als zu knapp, so daß neuere Versionen der DSPs hier ebenfalls auf 24 bit erweitert wurden.

Ein **MAC–**Ergebnis wird so abgelegt, daß der Akku-Teil **A0** bzw. **B0** die niederwertigsten 24 bit (**LSP** ≡ **L**east **S**ignificant **P**roduct) und der Teil **A1** bzw. **B1** die höherwertigen 24 bit (**MSP** ≡ **M**ost **S**ignificant **P**roduct) erhält. Überläufe werden in der Erweiterung **A2** bzw. **B2** abgelegt.

1.1.4.6 Das Programmsteuerwerk **CU**

Das Programmsteuerwerk (**CU** ≡ Control Unit) verfügt über drei 24 bit breite Kernregister, die den Programmadreßzähler (**PC** ≡ **P**rogram **C**ounter), die Statusinformation (im **SR** ≡ **S**tatus **R**egister) und die Betriebsart (im **OMR** ≡ **O**peration **M**ode **R**egister) beinhalten. Zwei zusätzliche Register mit den Bezeichnungen LC ≡ LOOP COUNTER und LA ≡ LOOP ADDRESS erlauben der Hardware, Schleifenoperationen ohne ’Software Overhead’ durchzuführen. Solche „DO-LOOPs“, die in der DSV sehr häufig sind, können mit der CU-Erweiterung ohne Zeitverzug ablaufen. Des weiteren verfügt die CU über 16 Stackregister mit 32 bit Breite und den zugehörigen Stackpointer.

Wegen der hohen Bedeutung in der DSV wird die Hardware DO-LOOP-Steuerung näher betrachtet. Für den Programmierer sind folgende Register aus Bild 1.20 bei Hardware DO-LOOPs von Bedeutung:

PC	Program Counter
LA	LOOP-Adresse (Adresse des letzten Befehls des Loops)
LC	LOOP-Counter (Anzahl der Durchläufe)

Der Stack hat 16 Plätze mit je 32bit und ist aufgeteilt in 2 Bänke SSH und SSL zu je 16bit. Hier können jeweils die 16 bit des PC und des SR untergebracht werden, so daß ein Unterprogramm oder ein Interrupt nur einen Stackplatz belegt.

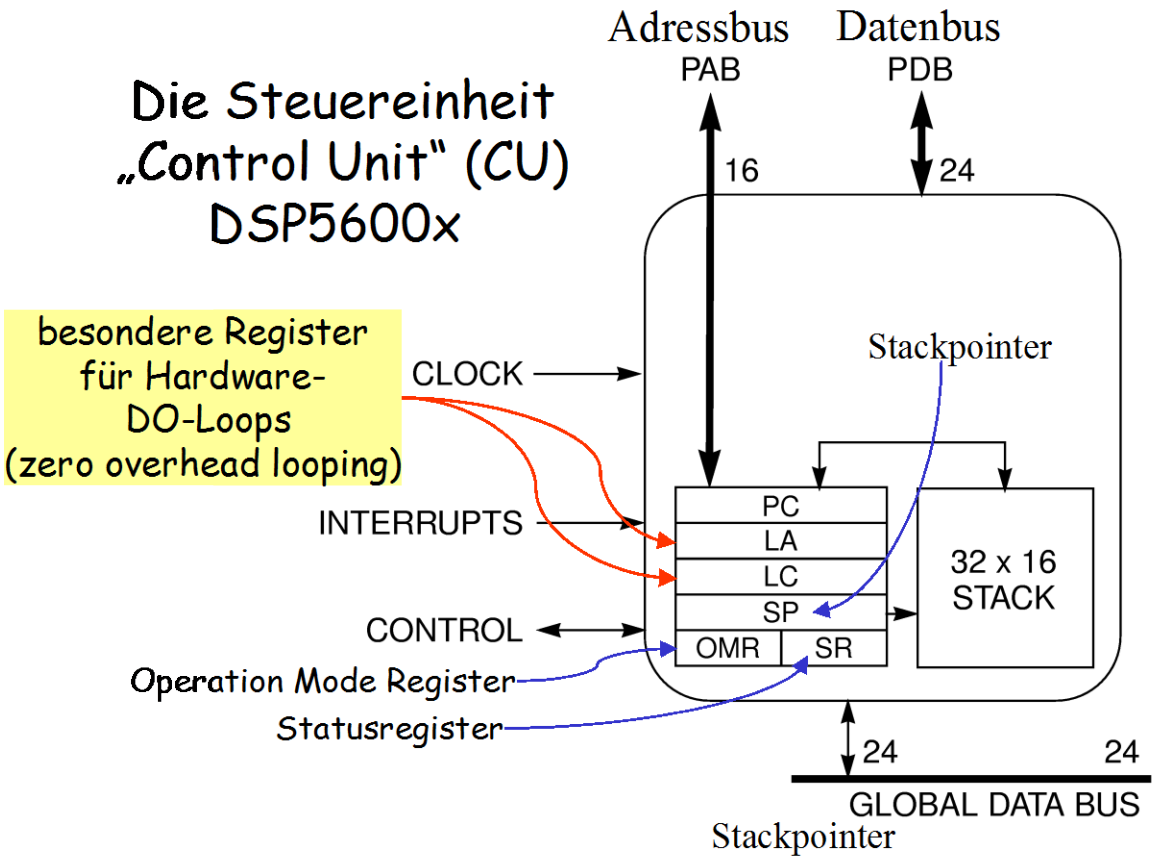


Bild 1.20: Übersicht über die Control Unit (CU) des DSP 5600x

Bei Hardware DO-LOOPs kommen die beiden neuen Register LA und LC hinzu, so daß zwei Plätze benötigt werden.

Eine typische Softwareschleife kann wie folgt aussehen:

```
FOR i=1 to N
        1. Befehl
        .
        .
        .
        letzter Befehl      ;bei jedem Befehl muß geprüft werden, ob das Ende der
                            Schleife schon erreicht ist
NEXT i                      ;hier muß getestet werden, ob i = N ist. Wenn nicht, muß i
                            erhöht und ein Rücksprung zum 1. Befehl ausgeführt werden
```

Dagegen kann ein Hardware-DO-LOOP, der z.B. eine Korrelationsfunktion gemäß Gleichung (1.1) mit N=1000 Abtastwerten realisiert, wie folgt im DSP programmiert werden:

```
        DO #1000, END1
        MAC X0,Y0,A    X:(R0)+,X0    Y:(R4)+,Y0
END1
```

das Korrelationsergebnis steht nach Ablauf im Akku A

Wie schon angedeutet, müssen beim Start eines Hardware-DO-LOOPs vier 16-bit-Register auf den Stack gebracht werden:

PC Program Counter	**SP** Stack Pointer	**SS** System Stack	**OMR** Operation Mode Reg.	**SR** Status Register	**LA** Loop Adresse	**LC** Loop Counter

⇑ ⇑ ⇑ ⇑

kommen auf den STACK bei DO-LOOPs

Details der Abarbeitung eines DO-LOOPs

DO #N, END1

⇓ ⇓

LC **LA** (letzter Befehl im LOOP)

PC ⇒ **Stack** ;markiert den LOOP-Anfang

SR ⇒ **Stack**

LA ⇒ **Stack** ;für Verschachtelung (Nesting)

LC ⇒ **Stack** ;für Verschachtelung (Nesting)

Folgende Schritte laufen während des LOOPs bei jedem OP-Code-Holen ab:

Vergleich: **PC ⇔ LA**

wenn **PC = LA** ;letzter LOOP-Befehl liegt vor

wenn **LC > 1** ;LC erniedrigen und weitermachen

LC:=LC-1

PC vom Stack holen und 1. LOOP-Befehl ausführen

Dieser Ablauf wiederholt sich so oft, bis

LC = 1

dann erfolgt noch ein genau **ein** Durchlauf

danach: LA, LC, SR vom Stack zurückholen

Programm läuft weiter bei PC:=LA+1

1.1.4.7 Zusammenfassung der DSP–Grundlagen

Aufgrund der parallel arbeitenden rechenfähigen Funktionseinheiten AGU und ALU und der Harvard-Architektur mit drei vollständigen ***separaten*** Bussystemen ist folgende Befehlsstruktur möglich, die wohl am besten den Leistungssprung vom Standard-Mikroprozessor zum DSP verdeutlicht.

Opcode	Operanden	X-Daten	Y-Daten
MAC	X0,Y0,A	X:(R0)+,X0	Y:(R4)+,Y0

Man sieht, daß parallel zur MAC–Operation zwei Datentransfers ausgeführt werden, die dafür sorgen, daß bei Wiederholung der Instruktion (typischerweise in einem LOOP) - nachdem die Pointer R0 und R4 jeweils um Eins erhöht worden sind – neue Daten in den Registern X0 undY0 vorhanden sind, die nun in die MAC–Operation einfließen.

Bild 1.21: Aufbau einer typischen DSP-Instruktion

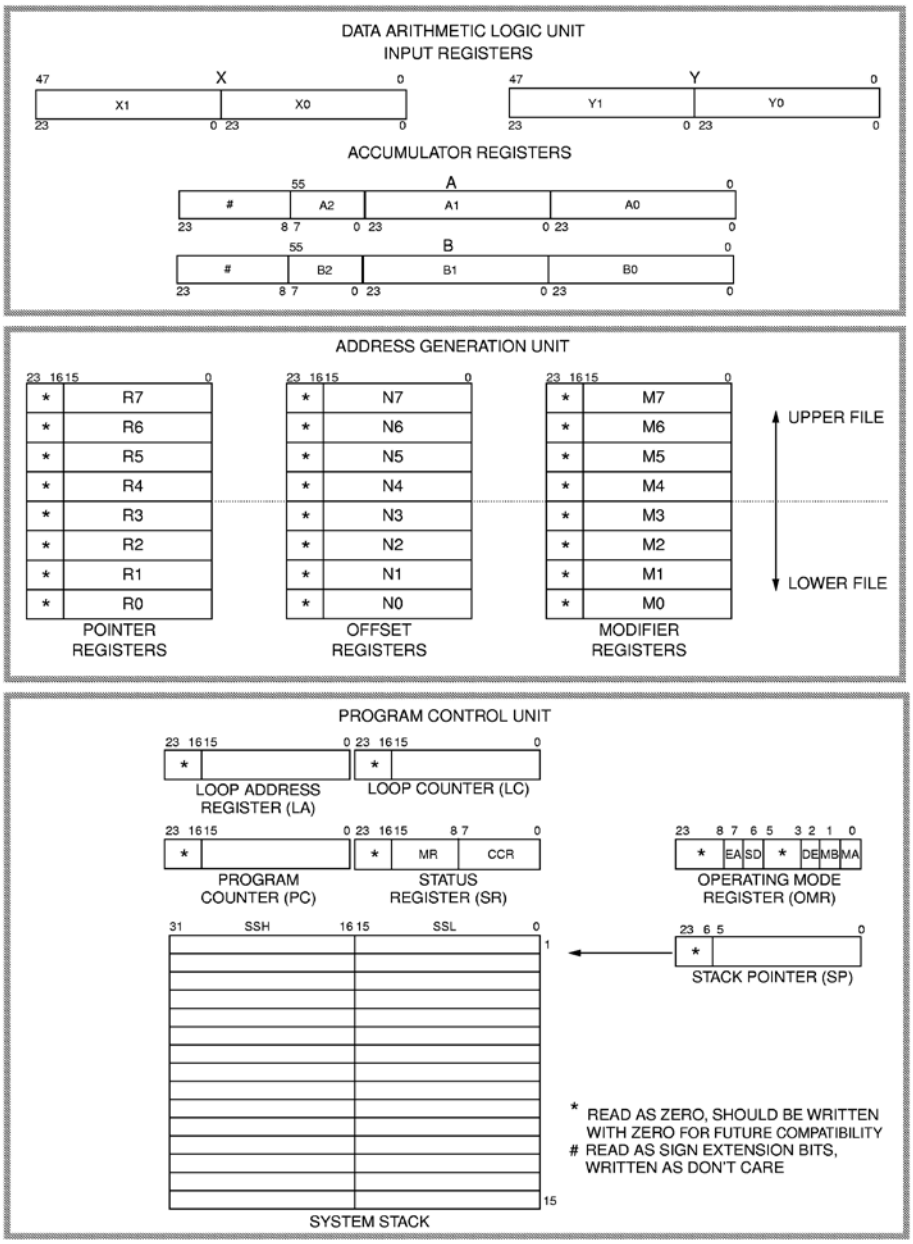


Bild 1.22: Das vollständige Programmiermodell des DSP5600x

Im Zusammenspiel mit '***Zero Overhead Looping***' lassen sich die typischen Algorithmen der DSV sehr effizient implementieren. Bei aller Komplexität stellt das Programmieren eines DSP einen mit Mikroprozessoren vertrauten Ingenieur keineswegs vor große Probleme, wie Bild 1.22 zusammenfassend verdeutlicht. Auch die im folgenden angegebene kurze Übersicht über den Befehlssatz des DSP56000 offenbart eine Reihe von Ähnlichkeiten mit Mikroprozessoren. Obwohl die Anzahl der DSP–Befehle meist kleiner ist als beim Standard–Mikroprozessor, muß aber an dieser Stelle darauf hingewiesen werden, daß die Befehlsstruktur im Detail beim DSP weitaus komplexer ist, so daß beim Programmieren der Blick ins Handbuch unerläßlich ist, vor allem wenn aufgrund harter Echtzeitbedingungen hardwarenah gearbeitet werden muß. Nicht ohne Grund befassen sich in der Regel etwa zwei Drittel eines DSP–Handbuchs mit der Befehlssatzbeschreibung.

Arithmetische Befehle

ABS	Absolute Value
ADC	Add Long with Carry
ADD	Addition
ADDL	Shift Left and Add
ADDR	Shift Right and Add
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
CLR	Clear an Operand
CMP	Compare
CMPM	Compare Magnitude
DEC	Decrement by One
DIV	Divide Iteration
INC	Increment by One
MAC	Signed Multiply-Accumulate
MACR	Signed Multiply-Accumulate and Round
MPY	Signed Multiply
MPYR	Signed Multiply and Round
NEG	Negate Accumulator
NORM	Normalize
RND	Round
SBC	Subtract Long with Carry
SUB	Subtract
SUBL	Shift Left and Subtract
SUBR	Shift Right and Subtract
Tcc	Transfer Conditionally
TFR	Transfer Data ALU Register
TST	Test an Operand

Logische Verknüpfungen/Schiebeoperationen

AND	Logical AND
ANDI	AND Immediate to Control Register
EOR	Logical Exclusive OR
LSL	Logical Shift Left
LSR	Logical Shift Right
NOT	Logical Complement
OR	Logical Inclusive OR
ORI	OR Immediate to Control Register
ROL	Rotate Left
ROR	Rotate Right

Befehle zur Bitmanipulation

BCLR Bit	Test and Clear
BSET Bit	Test and Set
BCHG Bit	Test and Change
BTST Bit	Test on Memory and Registers

Befehle zum LOOP–Aufbau

DO	Start Hardware Loop
ENDDO	Exit from Hardware Loop

Transferbefehle (Moves)

LUA	Load Updated Address
MOVE	Move Data Register
MOVEC	Move Control Register
MOVEM	Move Program Memory
MOVEP	Move Peripheral Data

Befehle zur Programmablaufsteuerung

II	Illegal Instruction
Jcc	Jump Conditionally
JMP	Jump
JCLR	Jump if Bit is Clear
JSET	Jump if Bit is Set
JScc	Jump to Subroutine Conditionally
JSCLR	Jump to Subroutine if Bit is Clear
JSR	Jump to Subroutine
JSSET	Jump to Subroutine if Bit is Set
NOP	No Operation
REP	Repeat Next Instruction
RESET	Reset On-Chip Peripheral Devices
RTI	Return from Interrupt
RTS	Return from Subroutine

STOP	Stop Processing (Low Power Stand-By)
SWI	Software Interrupt
WAIT	Wait for Interrupt (Low Power Stand-By)

1.1.4.8 Die 3-stufige Instruktions-Pipeline des DSP 56000 als Beispiel

Beispiel eines Programmausschnitts

I_1	**MACR**	**X0, Y1, A**	**X: (R0) +, X0**	**Y: (R4) +, Y1**
I_2	**CLR**	**A**	**X0, X: (R0) +**	**A, Y: (R4) -**
I_3	**MAC**	**X0, Y1, A**	**X: (R0) +, X0**	**Y: (R4) +, Y1**

Befehl holen FETCH Befehl dekodieren DECODE Befehl ausführen EXECUTION		I_1	I_2 I_1	I_3 I_2 I_1	I_4 I_3 I_2	I_5 I_4 I_3
parallele Operationen	Anfangs-bedingungen			↓ I_1 fertig	↓ I_2 fertig	↓ I_3 fertig
Adreß-Update (AGU)	R0=$0005 R4=$0008	– –	→ →	R0=5+1 R4=8+1	R0=6+1 R4=9-1	R0=7+1 R4=8+1
Befehlsausführung in der (Daten-ALU)	A: A2=$00 A1=$000066 A0=$000000 X0=$400000 Y1=$000077	– – – – – –	→ → → → → →	A: A2=$00 A1=$0000A2 A0=$000000 X0=$000005 Y1=$000008	A: A2=$00 A1=$000000 A0=$000000 X0=$000005 Y1=$000008	A: A2=$00 A1=$000000 A0=$000050 X0=$000007 Y1=$000008
X-Speicherinhalt bei Adresse $0005 $0006 $0007	DATEN $000005 $000006 $000007	 – – –	 → → →	 $000005 $000006 $000007	 $000005 $000005 $000007	 $000005 $000005 $000007
Y-Speicherinhalt bei Adresse $0008 $0009	DATEN $000008 $000009	 – –	 → →	 $000008 $000009	 $000008 $0000A2	 $000008 $0000A2

Aufgrund der Fraktaldarstellung gilt folgende Rechnung:

$$X0 \cdot Y1 = 5 \cdot 2^{-23} \cdot 8 \cdot 2^{-23} = 40 \cdot 2^{-46} = 80 \cdot 2^{-47} = 50\text{h} \cdot 2^{-47} \Rightarrow \text{A0=\$000050}$$

2 Interruptkonzepte

2.1 Interruptstrukturen in einfachen Mikroprozessoren und Mikrocontrollern

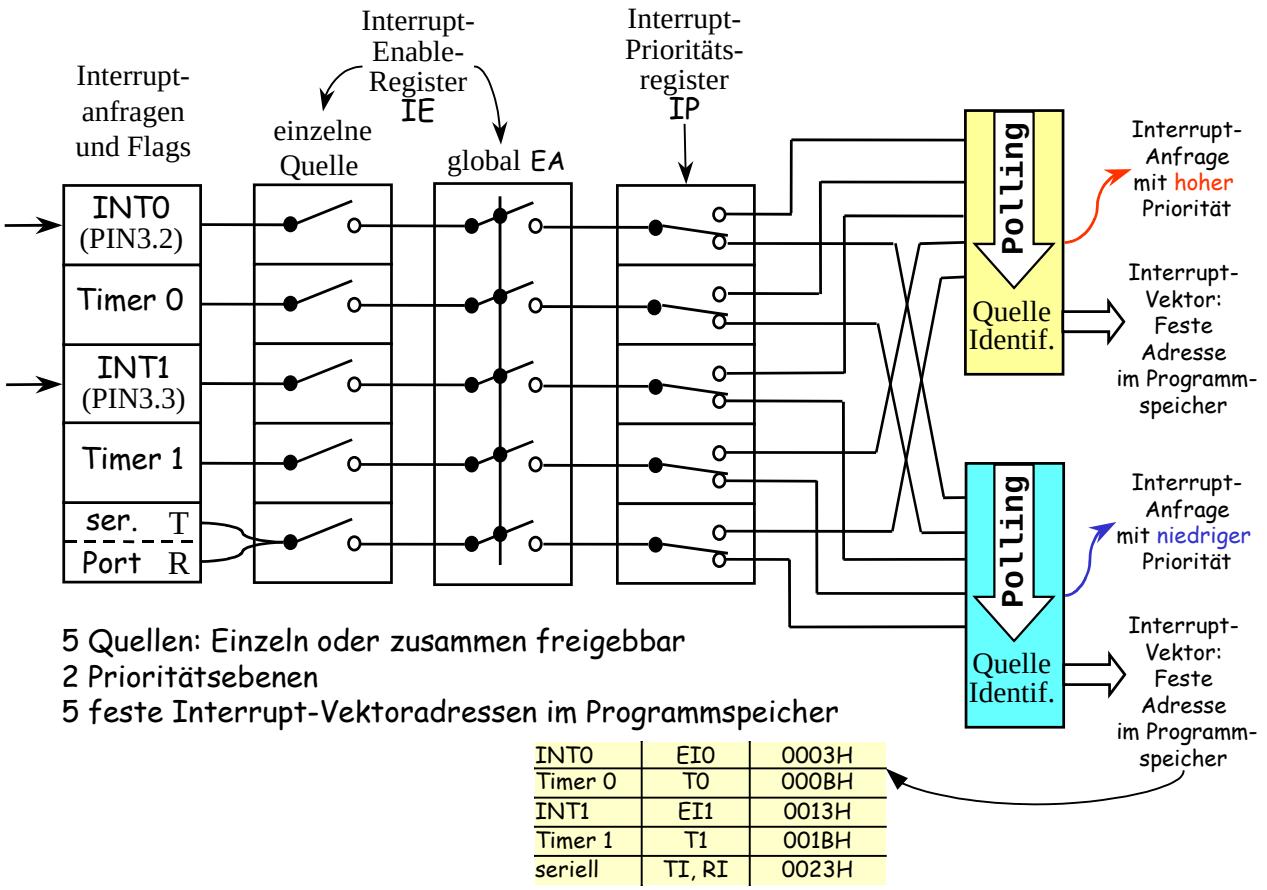


Bild 2.1: Das Interruptsystem eines einfachen Mikrocontrollers (8051)

Am Beginn einer Interruptbedienung in einem Standard-Mikroprozessor steht ein Sprung zu einer in der Regel fest zugewiesenen Vektoradresse. Die Interrupt-Vektoradressen sind in den meisten Fällen im unteren Bereich des Programmspeichers, z.B. ab 0002H zu finden. Zwischen den Interrupt-Vektoradressen liegen einige freie Speicherstellen, um zu ermöglichen, daß ein Sprung zur eigentlichen Bedienungsroutine hier programmiert werden kann.

Der Ablauf der Interruptbedienung (Interrupt-Serviceroutine ≡ ISR) ähnelt einem Unterprogrammaufruf, wobei der Stack in konventioneller Weise benutzt wird. Der wesentliche Unterschied ist, daß bei Interrupts eine ***hardwarebasierte*** Priorisierung erfolgt, d.h. Interrupts niederer Priorität können in der Abarbeitung unterbrochen werden, wenn ein Interrupt höherer Priorität auftritt. Am Ende einer ISR muß für ein ordnungsgemäßes Rücksetzen der Hardware gesorgt werden. Dazu steht in der Regel eine besonderer 'Return from Interrupt'-Befehl (z.B. RETI) zur Verfügung, der nicht mit dem Rückkehrbefehl aus einem Unterprogramm (wie RET) vertauscht werden darf.

Standard-Mikroprozessoren und Mikrocontroller haben meist nicht mehr als vier Prioritätsebenen. Die Zahl der Interruptquellen ist dagegen meist weit höher, z.B. größer als 30.

Zur Vermeidung von Konflikten bei der Bedienung von Interrupts gleicher Priorität wird das „Pollingprinzip" verwendet. Es handelt sich dabei um ein serielles Abfrage und Abarbeitungsverfahren, bei dem zu einem bestimmten Zeitpunkt in jedem Maschinenzyklus alle Interruptanfragen (Flags) getestet werden. Die Abarbeitung der registrierten Interrupts erfolgt dann in einer festgelegten Folge, wie das Beispiel in Bild 2.1 zeigt. Solange kein Interrupt höherer Priorität auftritt, wird eine begonnene ISR immer komplett abgearbeitet, auch wenn in den auf den Beginn folgenden Maschinezyklen Interruptanfragen auftauchen, die in der Pollingsequenz weiter oben stehen.

Allerdings wird die gemäß Pollingreihenfolge erstellte Abarbeitungstabelle in jedem Maschinezyklus der aktuellen Situation angepaßt.

2.1.1 Das 80C517/537 Interruptsystem

- 14 Quellen, 4 Prioritätsebenen
- je 7 externe und 7 interne Interrupts
- feste Interruptvektoren

Tabelle 2.1: Übersicht: Interruptquellen und -vektoren

Interruptanfrage-Flags	Interrupt-Vektoradresse	Interruptquelle
IE0	0003H	externer Interrupt 0
TF0	000BH	Timer 0 Überlauf
IE1	0013H	externer Interrupt 1
TF1	001BH	Timer 1 Überlauf
RI0/TI0	0023H	serielle Schnittstelle 0
TF2/EXF2	002BH	Timer 2 Überl./ext. Reload
IADC	0043H	A/D-Wandler
IEX2	004BH	externer Interrupt 2
IEX3	0053H	externer Interrupt 3
IEX4	005BH	externer Interrupt 4
IEX5	0063H	externer Interrupt 5
IEX6	006BH	externer Interrupt 6
RI1/TI1	0083H	serielle Schnittstelle 1
CTF	009BH	Compare-Timer-Überlauf

Tabelle 2.2: Kombinationen bei der Priorisierung. (Es gibt zwei Dreifachkombinationen und vier Zweifachkombinationen)

1. Tripel	externer Interrupt 0	serieller Interrupt 1	ADW-Interrupt
1. Paar	Timer 0 Interrupt	externer Interrupt 2	
2. Paar	externer Interrupt 1	externer Interrupt 3	
2. Tripel	Timer 1 Interrupt	Compare-Timer-Interrupt	externer Interrupt 4
3. Paar	serieller Interrupt 0	externer Interrupt 5	
4. Paar	Timer 2 Interrupt	externer Interrupt 6	

Jedes der Interrupt-Paare oder Tripel kann für sich einer von vier Prioritätsebenen zugeordnet werden. Das geschieht durch Setzen entsprechender Bits in den Spezialfunktionsregistern **IP0** und **IP1** (A9, B9)

A9			IP0.5	IP0.4	IP0.3	IP0.2	IP0.1	IP0.0
B9	-	-	IP1.5	IP1.4	IP1.3	IP1.2	IP1.1	IP1.0

BIT		Funktion
IP1.x	IP0.x	Prioritätsebene
0	0	0
0	1	1
1	0	2
1	1	3

Pollingsequenz

IE0 → RI1, TI1 → IDAC → TF0 → IEX2 → IE1 → IEX3 → TF1→ CTF → IEX4 → RI0, TI0 → IEX5 → TF2, EXF2 → IEX6

Ein Interrupt niederer Priorität kann von einem höherer Priorität unterbrochen werden. Auf gleicher Prioritätsebene ist gegenseitige Unterbrechung nicht möglich. Eine laufende Interruptbedienung auf der höchsten Prioritätsebene ist grundsätzlich nicht durch irgendeinen anderen Interrupt unterbrechbar. Falls zwei oder mehr Interruptanfragen auf verschiedenen Prioritätsebenen gleichzeitig auftreten, wird der Interrupt mit der höchsten Priorität zuerst bedient. Auf der gleichen Prioritätsebene bestimmt die obige Pollingsequenz die Abarbeitungs-Reihenfolge.

Interruptanfragen werden in Schritt 5, Phase 2 (S5P2) eines jeden Maschinenzyklus erfaßt. Die entsprechenden Flag-Bits (im SFR-Bereich) werden im folgenden Maschinenzyklus abgefragt. Wird ein gesetztes Flag gefunden, erfolgt ein Sprung zum zugehörigen Interruptvektor, wo die Bedienungsroutine zu programmieren ist. Dieser Ablauf erfolgt per Mikroprogramm unter folgenden Voraussetzungen:

1.) Es ist gerade kein Interrupt gleicher oder höherer Priorität in Bedienung.

2.) Der gerade abgelaufene Maschinenzyklus ist der letzte des momentan ab zuarbeitenden Befehls.

3.) Der momentan ablaufende Befehl ist nicht RETI oder ein Schreibzugriff auf eines der Register IEN0...3 oder IP0, IP1.

Bedingung 2.) sorgt dafür, daß jeder Befehl vollständig abgearbeitet wird, und Bedingung 3.) stellt sicher, daß eine vom Programmierer beabsichtigte Änderung des Interruptstatus von der CPU registriert wird.

Die Abfrage von Interrupt-Flags wird in jedem Maschinenzyklus durchgeführt. Relevant für die Pollingsequenz sind nur solche Flags, die in S5P2 des vorangehenden Maschinenzyklus aufgetreten waren. Kann ein Interrupt nicht sofort bedient werden, und wird sein zugehöriges Flag während der Wartezeit (Latenz) weggenommen, dann wird die Interruptanfrage ignoriert.

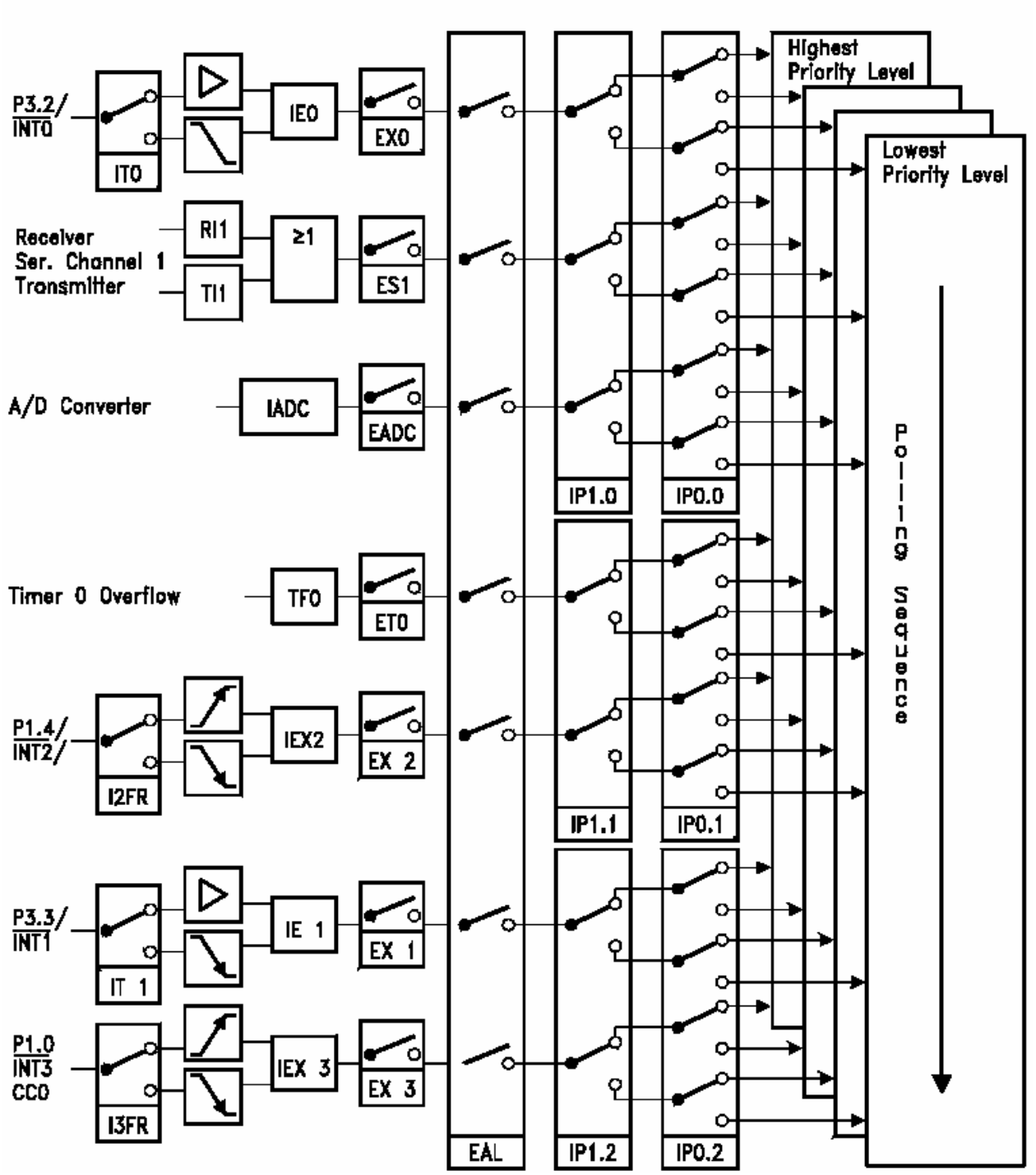


Bild 2.2: Übersicht über das Interruptsystem des 80C537 (Teil 1)

In den in Bild 2.2 und Bild 2.3 dargestellten Übersichten erkennt man links die verschiedenen Interruptquellen und ihre wichtigsten Eigenschaften. Es folgen die Flags, die die Interruptanfragen bewirken. In drei Fällen teilen sich zwei Interrupts dasselbe Flag: RI0&TI0, RI1&TI1 sowie TF2&EXF2. Nach den Freigabeschaltern für die einzelnen Quellen folgt der „Hauptschalter“ EAL, und dann die gruppenweise Prioritätszuordnung mittels der Bits IP0.x und IP1.x. Die Gruppierung hat zur Folge, daß alle Interrupts eines Tripels oder Paares nach Tabelle 2.2 stets auf der gleichen Prioritätsebene liegen.

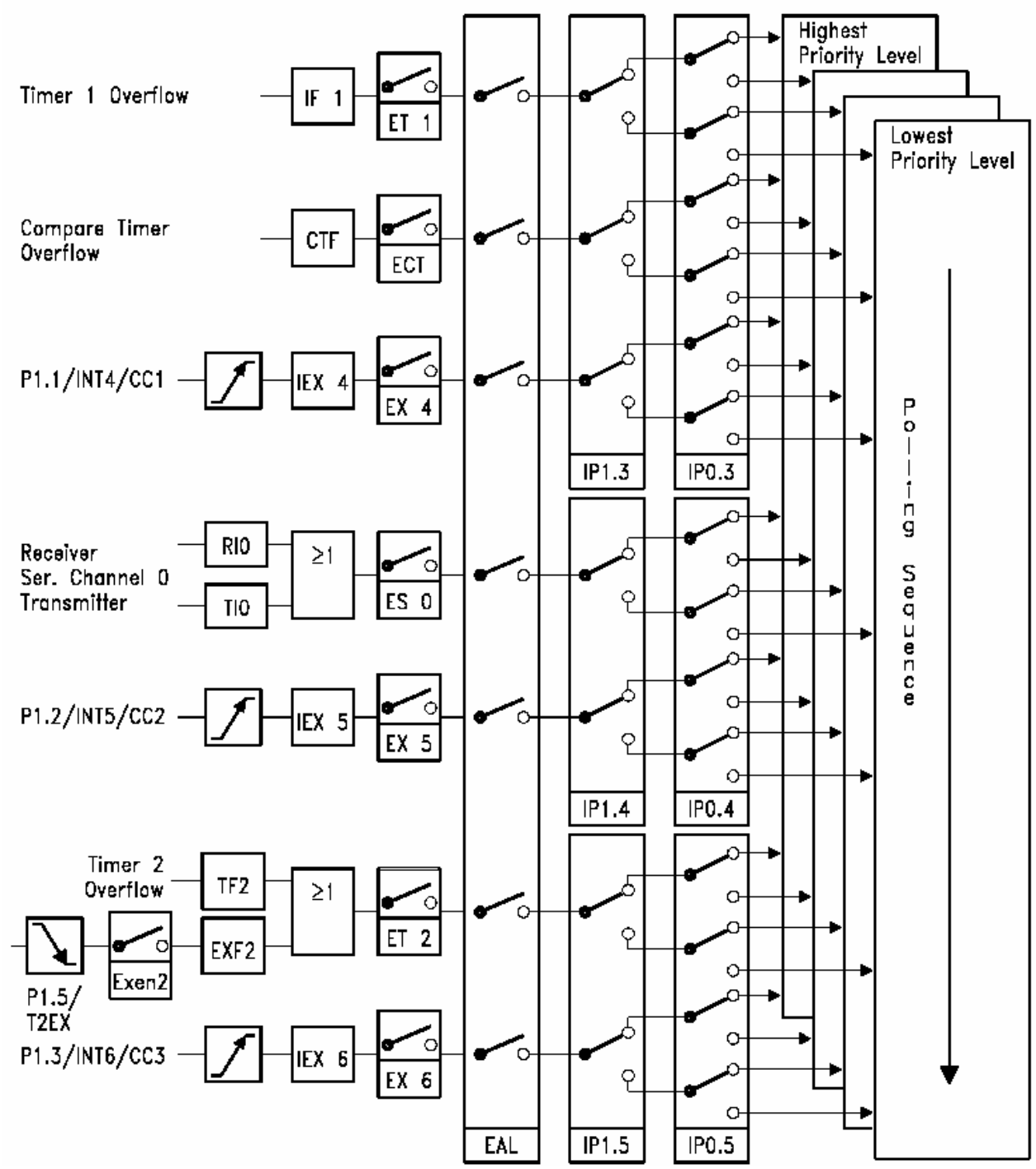


Bild 2.3: Übersicht über das Interruptsystem des 80C537 (Teil 2)

Das Ablaufdiagramm in Bild 2.4 zeigt ein Beispiel für eine typische Interrupt-Registrierung und –Bedienung.

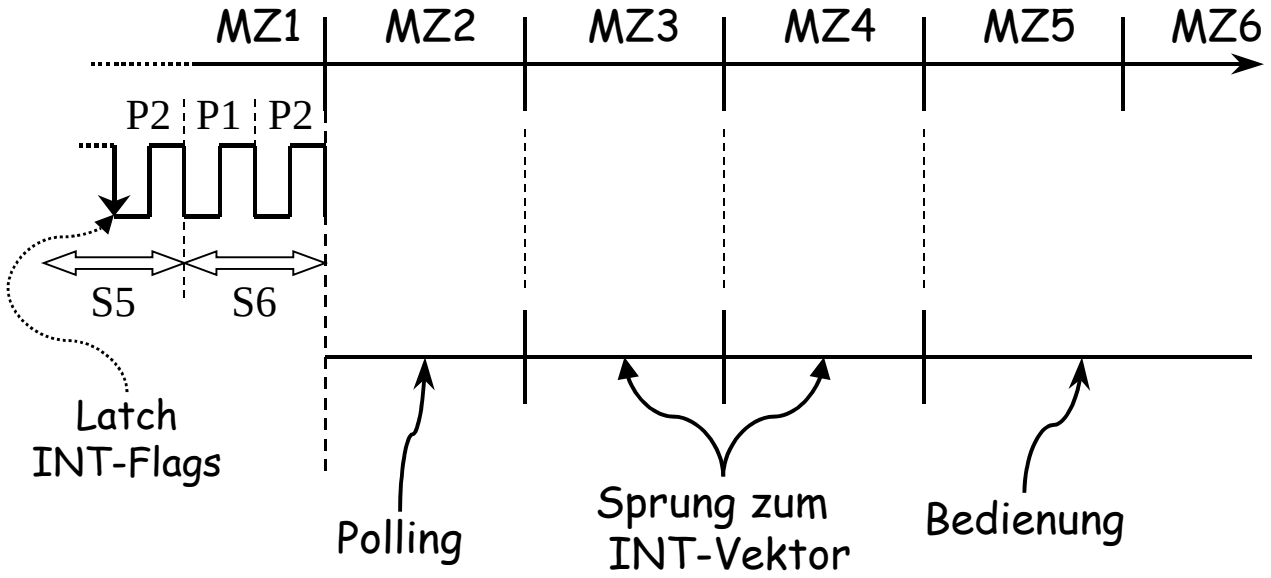


Bild 2.4: Interrupt-Registrierung und –Bedienung im 80C537

Bemerkung 1:

Falls ein Interrupt höherer Priorität vor S5P2 im MZ3 registriert wird, dann wird entsprechend den oben formulierten Regeln die Bedienung dieses Interrupts in den Zyklen MZ5 und MZ6 eingeleitet, ohne daß auch nur ein Befehl des Interrupts niederer Priorität abgearbeitet wurde.

Die Bedienung eines Interrupts besteht grundsätzlich in einer Verzweigung des Programms zum entsprechenden Interruptvektor. Dabei wird wie bei einem Unterprogrammaufruf der Inhalt des PC (16 Bit) auf den Stack geschoben und die Interrupt-Vektoradresse in den PC geladen.

Bemerkung 2:

Das Programmstatuswort (PSW) wird nicht automatisch auf den Stack gerettet!

Die Interruptbedienung erfolgt ab dem zugehörigen Vektor, bis ein RETI–Befehl erreicht wird. RETI beendet die Routine, holt zwei Byte vom Stack und speichert sie in den PC zurück, so daß das Programm an der Unterbrechungsstelle weiterläuft.

Der RETI–Befehl am Ende einer Interruptbedienung ist wichtig, weil nur dieser Befehl die Interrupthardware in den korrekten Status zurückversetzt. Mit RET (Unterprogramm-Rücksprung) könnte man auch die Interruptroutine verlassen, aber die Hardware würde dies nicht registrieren. Eine mögliche Folge wäre, daß keine weiteren Interrupts mehr zur Abarbeitung kämen, wenn z.B. der gerade abgearbeitete Interrupt die höchste Priorität gehabt hätte.

2.1.2 Die Interruptstruktur im 16bit-Mikrocontroller 80C166 mit Pipelining

Der 80C166 ist ein langjährig bewährter 16bit-Mikrocontroller, der auf zeitkritische und rechenintensive Steuer-, Regelungs- und Signalverarbeitungsaufgaben vor allem für KFZ–Anwendungen hin konzipiert wurde. Es war eine Neuentwicklung auf 16 bit-Ebene, ohne Abwärtskompatibilität nach 8 bit. Bewährte 8 bit-Hard- und Software-Konzepte aus der 8051-Architektur haben jedoch das Design stark beeinflußt.

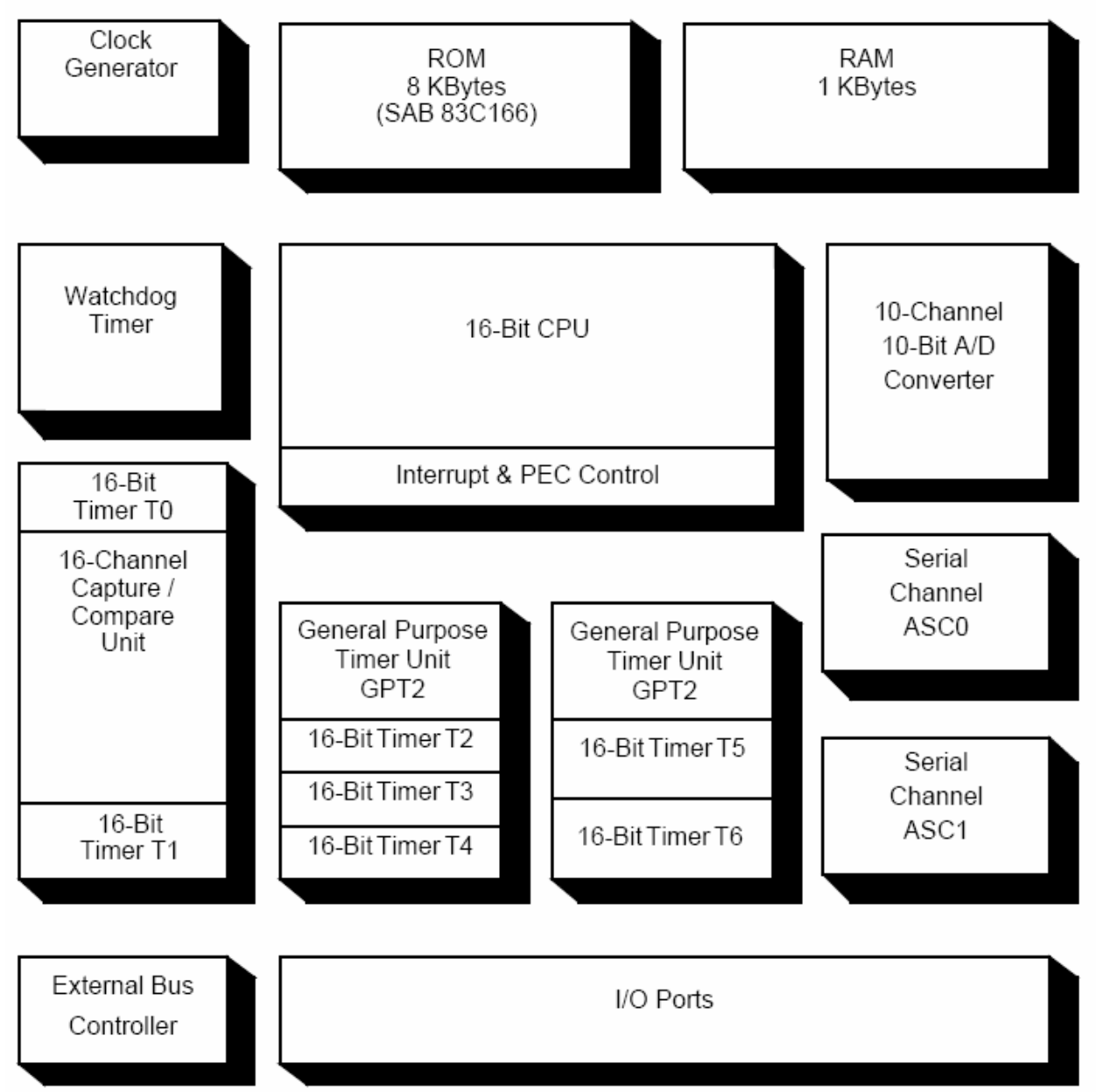


Bild 2.5: Übersichtsblockschaltbild des 80C166

Haupteigenschaften

CPU

- 1 Maschinenzyklus dauert 100ns ⇒ 10 MIPS
- 16bit·16bit Multiplikation in 500ns (kein Parallelmultiplizierer)
- 32bit/16bit Division in 1µs
- über 4 GByte linearer Adreßraum , gemeinsam für OP-Code und Daten (von Neumann Architektur)

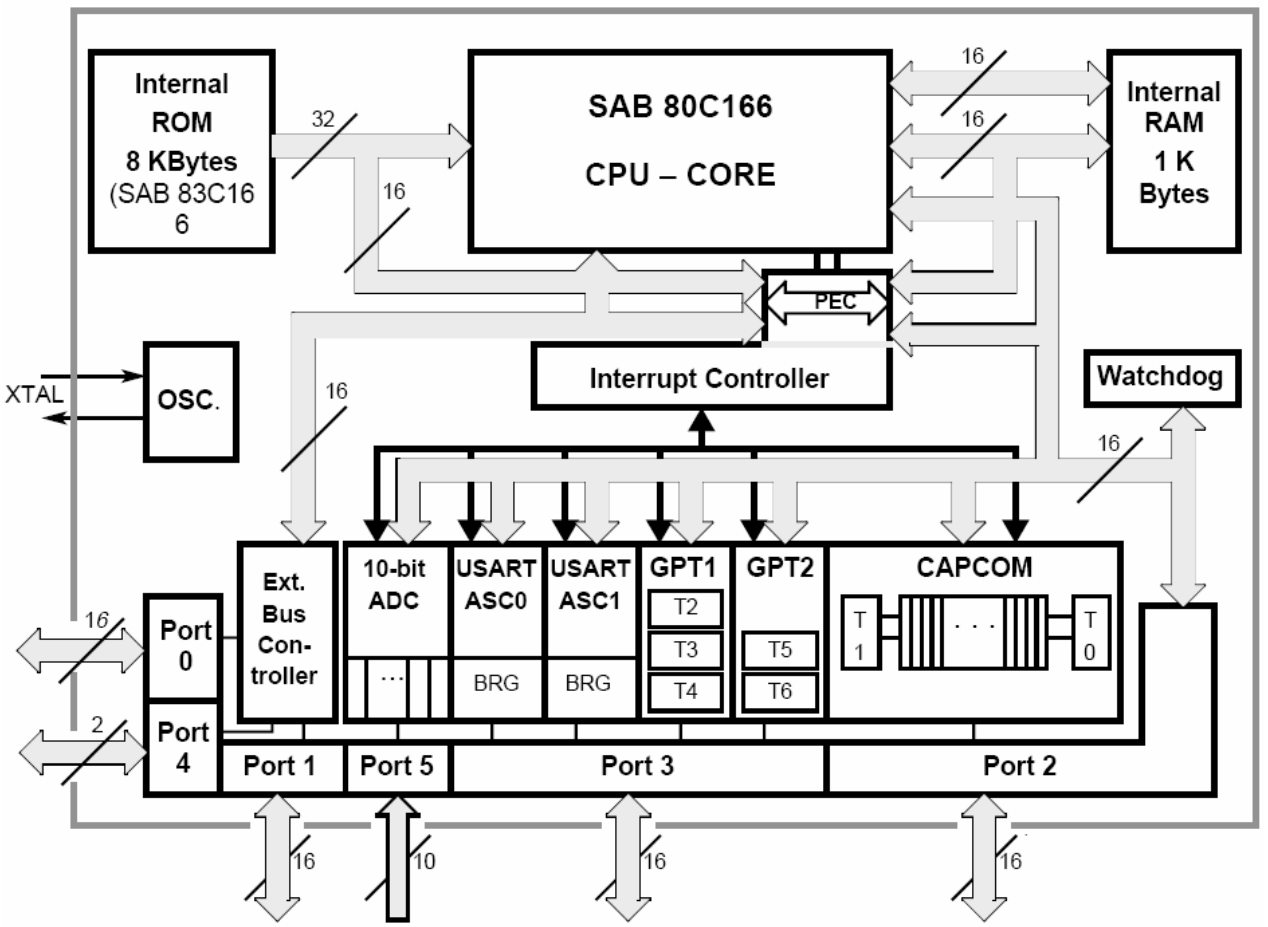


Bild 2.6: Detailliertes Blockschaltbild des 80C166

integrierter Speicher

- 1 kByte RAM
- 8 kByte ROM (83C166)

Interruptsystem

- 32 Interruptquellen mit festen Interruptvektoren
- 16 Prioritätsebenen
- typische Latenzzeiten 300...500ns
- 8-kanaliger ***'Peripheral-Event-Controller'*** (PEC). Der PEC stellt eine besondere Art der Interruptverarbeitung ohne Softwareoverhead dar (ähnlich den Fast Interrupts beim DSP)

10-kanaliger 10bit-A/D-Wandler

- Wandelzeit: 9,75µs

Umfangreiches, schnelles Timer-System

- Universal-Timer-System GPT1: drei 16 bit-Timer/Counter (400ns Auflösung)
- Universal-Timer-System GPT2: zwei 16 bit-Timer/Counter (200ns Auflösung)
- 16-kanalige Capture/Compare-Einheit mit zwei separaten Zeitbasen (400ns Auflösung) (erlaubt flexible PWM-Programmierung)
- Watchdog-Timer

zwei serielle Schnittstellen

- für synchronen oder asynchronen Betrieb mit unabhängigen Baudratengeneratoren

umfangreiche Einzelbitverarbeitung

76 Ein/Ausgabeleitungen

- individuell bitadressierbar
- als Eingang $R_e \rightarrow \infty$, mit Schmitt-Trigger-Verhalten

Besonderheiten

- ein Maschinenzyklus pro Befehl mit Ausnahme der MUL/DIV- und Verzweigungsbefehle
- Instruktions-Pipelining in einer 4-stufigen Pipeline

⇒ FETCH ein Befehl wird aus dem internen ROM bzw. RAM oder aus einem externen Speicher geholt

⇒ DECODE der geholte Befehl wird dekodiert und zugehörige Operanden werden geholt

⇒ EXECUTE der Befehl wird ausgeführt

⇒ WRITE BACK Ergebnisse werden an die im Befehl spezifizierten Speicherstellen geschrieben

Ohne Pipelining würde jeder Befehl mindestens 4 Maschinenzyklen dauern!

Tabelle 2.3: Pipelining Schema des 80C166

FETCH	B1	B2	B3	B4	B5	B6
DECODE		B1	B2	B3	B4	B5
EXECUTE			B1	B2	B3	B4
WRITE BACK				B1	B2	B3
	→	1 Zyklus	←		Zeit →	

Gesamtphilosophie:

⇒ Symbiose aus RISC- und CISC–Konzepten

⇒ nur wenige und häufig gebrauchte komplexe Befehle

⇒ keine Zwischenergebnisse (erfordern temporäre Speicherung)

⇒ keine komplizierten Adressierungsarten

⇒ „Ein-Wort-Format“ für die am häufigsten benutzten Befehle

⇒ echtzeitfreundliches Interruptkonzept

⇒ PEC–Interrupts können mit einem eingeschobenen Maschinenzyklus bedient werden ohne Verzweigung zu einer Bedienungsroutine

⇒ zahlreiche Registerbänke (16 Arbeitsregister), die beliebig im RAM plaziert sein können (Bankumschaltung ohne Verzug erfolgt im aktuellen Maschinenzyklus)

⇒ Mehrzyklen-Befehle sind unterbrechbar

Der 80C166 arbeitet ohne Mikrocode. Die Befehlsdekodierung wird über PLA-Strukturen (Programmable Logic Array) durchgeführt ⇒ unmittelbares Ansteuern der Hardware

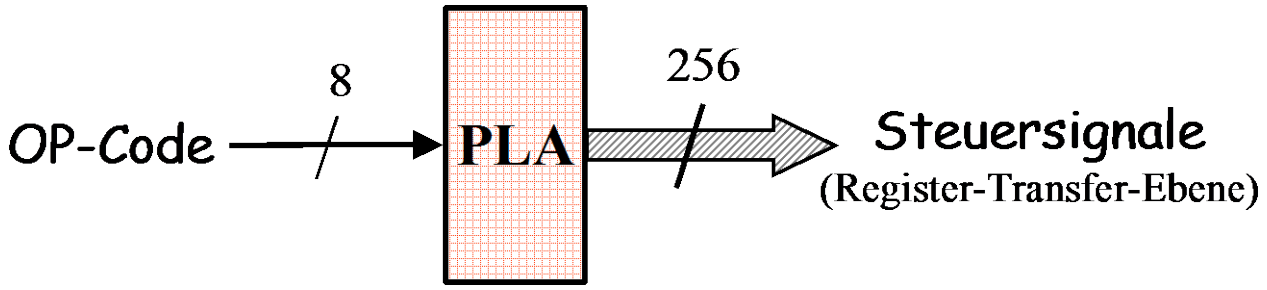


Bild 2.7: Beispiel einer einfachen PLA–basierten Befehlsdekodierung

Bei der Befehlsdekodierung mittels PLA werden die Steuersignale, die in nachfolgenden Pipelinestufen benötigt werden, in Registern gespeichert. Die Pipeline kann deshalb angehalten werden, z.B. um ***'Wait-States'*** für den Zugriff auf langsame Peripheriekomponenten (Speicher) einzufügen. Mehrzyklenbefehle werden durch Befehlsinjektion ausgeführt, wobei einfache Finite-State-Machines zur Modifikation der Steuersignale vorhanden sind.

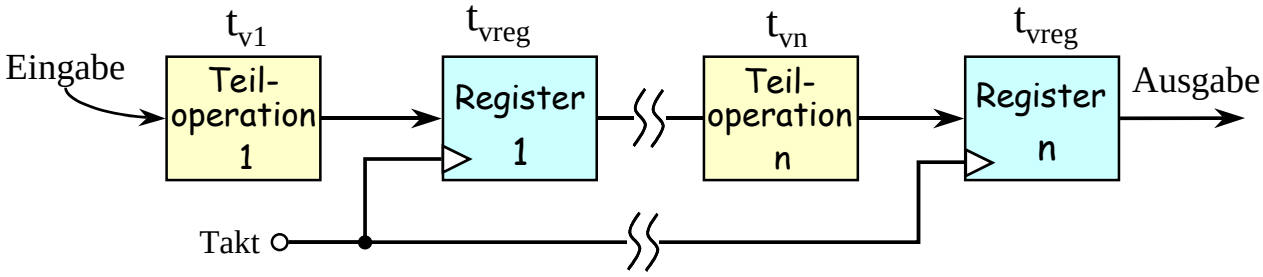


Bild 2.8: Aufspaltung umfangreicher Operationen in Teiloperationen durch Einfügen von Registern

Vorteile des Pipelining–Prinzips:

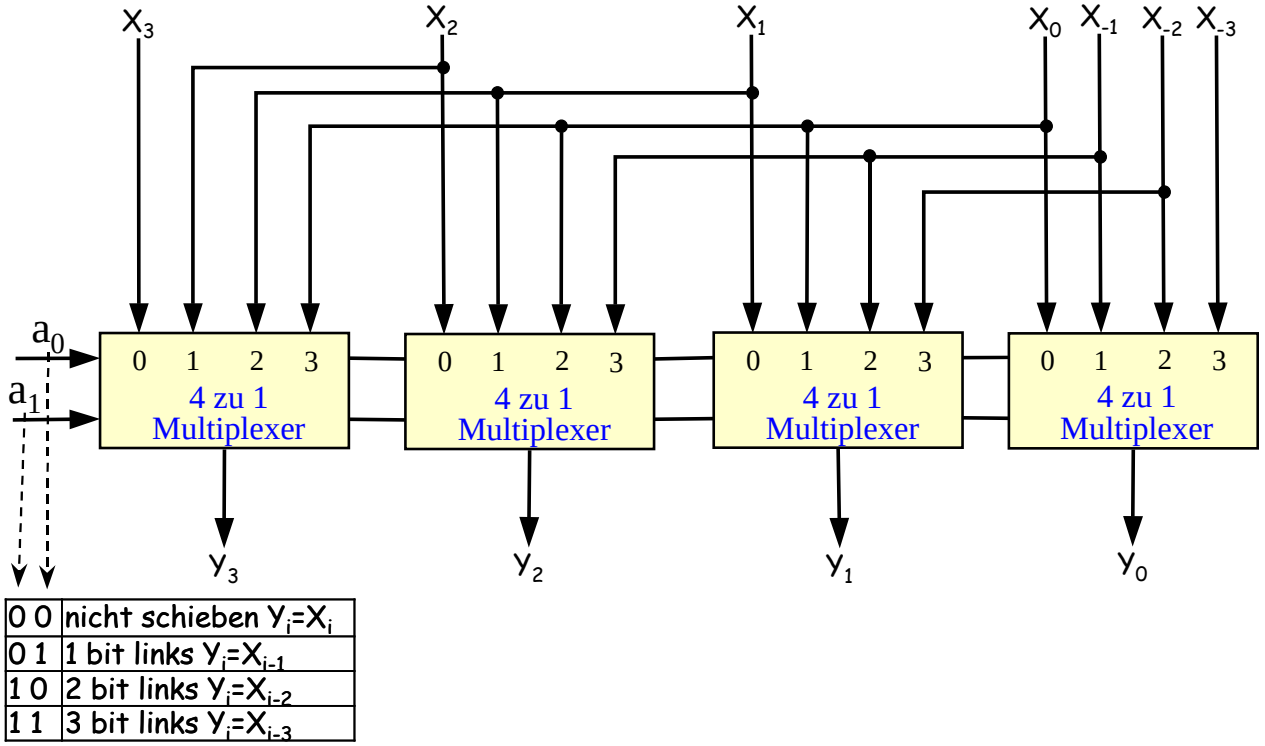
Arbeitsfrequenz: $$f_z = \frac{1}{t_{vi\max} + t_{vreg}} \qquad (2.1)$$

Beispiel:

$$t_{v1} = 10\text{ns},\ \ t_{v2} = 13\text{ns},\ t_{v3} \ldots t_{vn} = \underbrace{15\text{ns}}_{\max.};\ \ t_{vreg} = 5\text{ns}$$

⇒ unabhängig von der Zahl der Pipeline-Stufen ist $f_z = \frac{1}{20ns} = 50\text{MHz}$

ohne Pipelining wäre $$f_z = \frac{1}{t_{v1} + t_{v2} + \ldots t_{vn}}$$



0 0	nicht schieben $Y_i=X_i$
0 1	1 bit links $Y_i=X_{i-1}$
1 0	2 bit links $Y_i=X_{i-2}$
1 1	3 bit links $Y_i=X_{i-3}$

Bild 2.9: Der ***Barrel-Shifter*** dient zur Ausführung mehrerer Schiebeoperationen in einem Maschinenzyklus

Besonderheiten des internen RAMs

enthält: 'general purpose' Registersatz GPR (32 byte)
(R0...R7 sind byteorganisiert, R8...R15 sind wortorganisiert)

alle GPR und der obere RAM-Bereich sind bitadressierbar

PEC - Quell- und Ziel-Pointer: 8 Paare mit je 16bit Länge

⇒ jedes Paar ist einem bestimmten PEC-Kanal zugeordnet

⇒ **Zweck:** Schneller Datentransfer ohne Interruptbedienung

2.1.2.1 Besonderheiten bei Interrupt–Verarbeitung im 80C166

Neben der gewöhnlichen software-orientierten Interruptbedienung mit 16 Prioritätsebenen erlaubt der 80C166 eine erheblich schnellere Alternative durch den integrierten „Peripheral-Event-Controller" (PEC). Bei einer Interruptanfrage erlaubt der PEC den Transfer eines Datenbytes oder –worts zwischen zwei beliebigen Stellen im Segment 0 des Speichers. Dabei wird der normale Programmablauf nicht unterbrochen, sondern um einen Maschinenzyklus verzögert. Der PEC-Service ist sehr hoch priorisiert, er wird nur für die Prioritätsebenen 14 und 15 aktiviert.

Es gibt kein „Polling" bei gleich priorisierten Interrupts, sondern es muß vom Programmierer eine ***'Gruppenpriorität'*** vergeben werden, wenn mehrere Interruptquellen der gleichen Prioritätsebene zugewiesen sind. Dabei dürfen maximal 4 (eine Gruppe) der 32 Interruptquellen auf dieselbe Prioritätsebene gelegt werden. Mit 2 bit ist dann die Gruppenpriorität eindeutig definiert. Jede Interruptquelle hat ein eigenes 8 bit breites Steuerregister mit folgendem Aufbau:

7	6	5	4	3	2	1	0
Int.-Flag	Enable	Priorität [0...F]				Gruppenpriorität	

Die CPU kopiert die aktuelle Interruptpriorität ins Programmstatuswort (PSW). Bei ***Reset*** wird Priorität Null eingetragen. In jedem Maschinenzyklus werden alle Interruptquellen abgefragt. Diejenige mit höchster Priorität, die bedient wird, hinterläßt für die Dauer ihrer Bedienung ihre Priorität im PSW. Nach Ablauf der Bedienungsroutine wird der vorhergehende Status vom Stack ins PSW zurückgeholt. So ist gewährleistet, daß stets nur Interruptanfragen höherer Priorität als der gerade in Abarbeitung befindliche Interrupt eine Möglichkeit zur Unterbrechung erhalten.

Ein PEC–Datentransfer veranlaßt keine Änderung des Prioritätseintrages im PSW, da nur ein einziger Maschinenzyklus dafür benötigt wird.

Tabelle 2.4: Interruptquellen und zugeordnete Vektoren

Interruptquelle oder PEC Service Request	Request Flag	Enable Flag	Interrupt-vektor	Vekto-radresse	Trap-Nummer
CAPCOM Register 0	CC0IR	CC0IE	CC0INT	40h	10h
CAPCOM Register 1	CC1IR	CC1IE	CC1INT	44h	11h
CAPCOM Register 2	CC2IR	CC2IE	CC2INT	48h	12h
CAPCOM Register 3	CC3IR	CC3IE	CC3INT	4Ch	13h
CAPCOM Register 4	CC4IR	CC4IE	CC4INT	50h	14h
CAPCOM Register 5	CC5IR	CC5IE	CC5INT	54h	15h
CAPCOM Register 6	CC6IR	CC6IE	CC6INT	58h	16h
CAPCOM Register 7	CC7IR	CC7IE	CC7INT	5Ch	17h
CAPCOM Register 8	CC8IR	CC8IE	CC8INT	60h	18h
CAPCOM Register 9	CC9IR	CC9IE	CC9INT	64h	19h
CAPCOM Register 10	CC10IR	CC10IE	CC10INT	68h	1Ah
CAPCOM Register 11	CC11IR	CC11IE	CC11INT	6Ch	1Bh
CAPCOM Register 12	CC12IR	CC12IE	CC12INT	70h	1Ch
CAPCOM Register 13	CC13IR	CC13IE	CC13INT	74h	1Dh
CAPCOM Register 14	CC14IR	CC14IE	CC14INT	78h	1Eh
CAPCOM Register 15	CC15IR	CC15IE	CC15INT	7Ch	1Fh
CAPCOM Timer 0	T0IR	T0IE	T0INT	80h	20h
CAPCOM Timer 1	T1IR	T1IE	T1INT	84h	21h
GPT1 Timer 2	T2IR	T2IE	T2INT	88h	22h
GPT1 Timer 3	T3IR	T3IE	T3INT	8Ch	23h
GPT1 Timer 4	T4IR	T4IE	T4INT	90h	24h
GPT2 Timer 5	T5IR	T5IE	T5INT	94h	25h
GPT2 Timer 6	T6IR	T6IE	T6INT	98h	26h
GPT2 CAPREL Register	CRIR	CRIE	CRINT	9Ch	27h
A/D Conversion Complete	ADCIR	ADCIE	ADCINT	A0h	28h
A/D Overrun Error	ADEIR	ADEIE	ADEINT	A4h	29h
Serial Channel 0 Transmit	S0TIR	S0TIE	S0TINT	A8h	2Ah
Serial Channel 0 Receive	S0RIR	S0RIE	S0RINT	ACh	2Bh
Serial Channel 0 Error	S0EIR	S0EIE	S0EINT	B0h	2Ch
Serial Channel 1 Transmit	S1TIR	S1TIE	S1TNT	B4h	2Dh
Serial Channel 1 Receive	S1RIR	S1RIE	S1RINT	B8h	2Eh
Serial Channel 1 Error	S1EIR	S1EIE	S1EINT	BCh	2Fh

Tabelle 2.5: Reset- und Trap–Vektoradressen

Exception Condition	Trap Flag	Trap Vector	Vector Location	Trap Number	Trap Priority
Reset Functions:					
Hardware Reset	-	RESET	0h	0h	III
Software Reset	-	RESET	0h	0h	III
Watchdog Timer Overflow	-	RESET	0h	0h	III
Class A Hardware Traps:					
Non-Maskable Interrupt	NMI	NMITRAP	08h	2h	II
Stack Overflow	STKOF	STOTRAP	10h	4h	II
Stack Underflow	STKUF	STUTRAP	18h	6h	II
Class B Hardware Traps:					
Undefined Opcode	UNDOPC	BTRAP	28h	Ah	I
Protected Instruction Fault	PRTFLT	BTRAP	28h	Ah	I
Illegal Word Operand Access	ILLOPA	BTRAP	28h	Ah	I
Illegal Instruction Access	ILLINA	BTRAP	28h	Ah	I
Illegal External Bus Access	ULLBUS	BTRAP	28h	Ah	I
Reserved			[2Ch-3Ch]	[Bh-Fh]	
Software Traps:					
TRAP Instruction			Any [0h-1FCh] in steps of 4H	Any [0h-7Fh]	Current CPU Priority

2.1.2.2 Wie funktioniert der ***PEC*** und was ist ’Cycle-Stealing’?

Es gibt 8 ***PEC-Kanäle***, die jeweils über ein Steuerregister PECC0...PECC7, einen Quellenzeiger SRCP0...SRCP7 sowie einen Zielzeiger DSTP0...DSTP7 verfügen.

Ein Steuerregister ***PECCi*** hat folgenden Aufbau:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					INC		BWT	C O U N T							

Symbol	Bedeutung	
COUNT	FF	kontinuierlicher Transfer, kein Dekrement
	FE ≥ COUNT ≥ 1	Dekrement bei jedem Transfer
	00	Interruptanfrage an CPU
BWT	„0“ Worttransfer; „1“ Bytetransfer	
INC	00 ⇒ kein INC; 01 ⇒ Zielzeiger-INC; 10 ⇒ Quellzeiger-INC	

Es gilt folgende Zuordnung der PEC-Kanäle 0...7 zu den Interruptquellen:

Nur die Prioritätsebenen 14 (1110) und 15 (1111) erlauben PEC-Service. Auf jeder dieser Ebenen dürfen 4 Interruptquellen per Gruppenpriorität eingeordnet werden. Es ergibt sich so folgende Zuordnung für den Fall, daß im entsprechenden Steuerregister PECCi der Wert von COUNT ≠ 0 ist:

Tabelle 2.6: Zuordnung der PEC-Kanäle 0...7 zu den Interruptquellen

Interruptpriorität	Gruppenpriorität	PEC-Kanal-Nr.
(14) 1110	00	0
(14) 1110	01	1
(14) 1110	10	2
(14) 1110	11	3
(15) 1111	00	4
(15) 1111	01	5
(15) 1111	10	6
(15) 1111	11	7

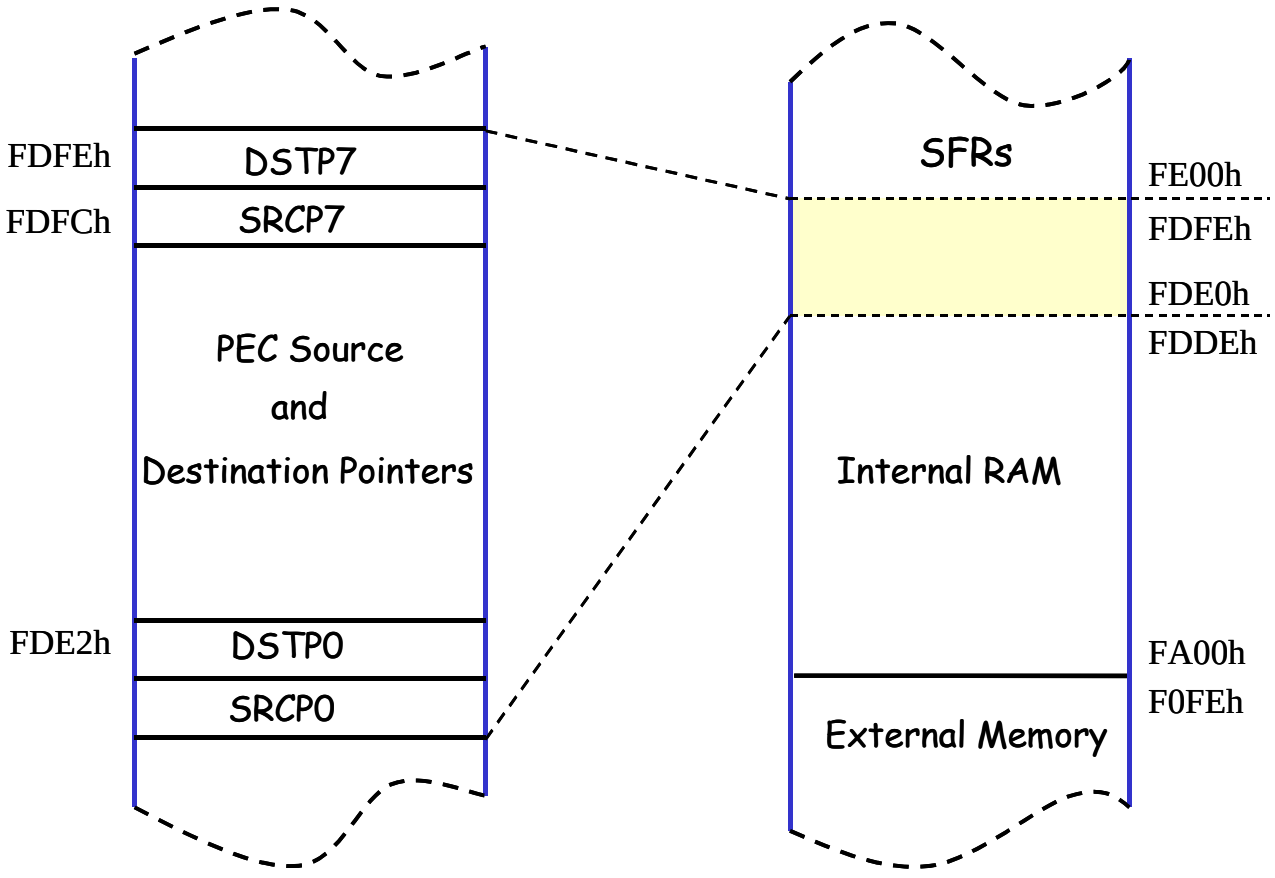


Bild 2.10: Lage der PEC-Pointer im Speicherbereich

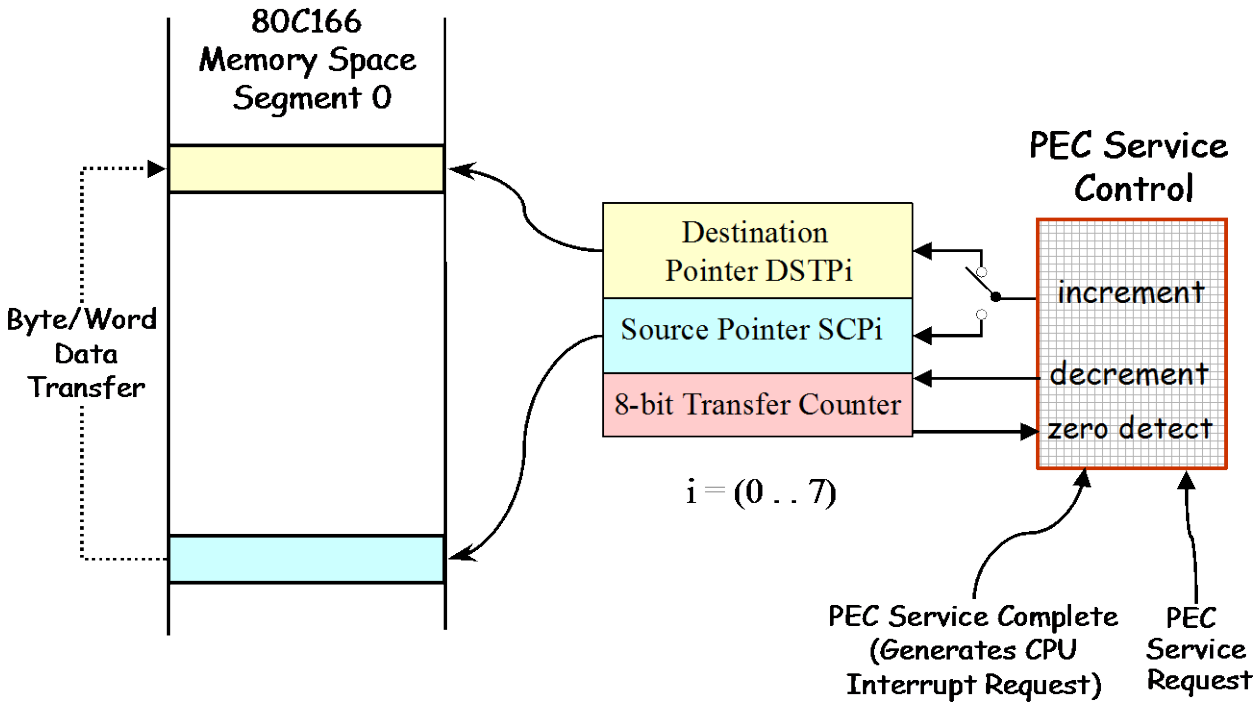


Bild 2.11: Beispiel zur Erläuterung der PEC-Verarbeitung

Funktion der Pipeline bei Verzweigungen und bei der PEC-Verarbeitung

Leerzyklus einfügen ↓ Verzweigungsziel einlesen ⇓ ⇓

FETCH	Verzw.	Bn+2	V.-Ziel	Bz+1	Bz+2	Bz+3
DECODE	Bn	Verzw.	EINFÜG	V.-Ziel	Bz+1	Bz+2
EXECUTE	...	Bn	Verzw.	EINFÜG	V.-Ziel	Bz+1
WRITE BACK	...	...	Bn	Verzw.	EINFÜG	V.-Ziel

→ 1 Zyklus ← Zeit →

2.1.3 Interruptverarbeitung im DSP am Beispiel DSP5600x

- 4 externe und 16 interne Interrupt Quellen
- 4 Prioritätsebenen IPL0...3 (Interrupt Priority Level)
- Level 0, 1, 2 maskierbar; Level 3 nicht!
- Level-3-Interrupts werden immer bedient
- 2 bit (Interrupt Mask I0, I1) im Statusregister geben den aktuellen Interruptstatus wieder
- Es gibt 32 feste Interruptvektoren auf den unteren 64 Speicherplätzen im Programmspeicherbereich 0...3Fh. Dort stehen je 2 Worte á 24 bit für eine Standard-Serviceroutine zur Verfügung.

Es werden ***fast*** und long Interrupts unterschieden:

- ein ***fast Interrupt*** wird mit maximal zwei an der zugehörigen Vektoradresse gespeicherten Befehlen vollständig bedient; der Stack wird dabei nicht benutzt.
- bei einem long Interrupt wird mit einem Sprungbefehl an der zugehörigen Vektoradresse in der Regel zu einem Unterprogramm verzweigt. Dabei wird der Stack wie in einem konventionellen Mikroprozessor benutzt.

Interrupt Starting Address	IPL	Interrupt Source
$0000	3	Hardware $\overline{\text{RESET}}$
$0002	3	Stack Error
$0004	3	Trace
$0006	3	SWI
$0008	0 - 2	$\overline{\text{IRQA}}$
$000A	0 - 2	$\overline{\text{IRQB}}$
$000C	0 - 2	SSI Receive Data
$000E	0 - 2	SSI Receive Data With Exception Status
$0010	0 - 2	SSI Trasmit Data
$0012	0 - 2	SSI Transmit Data with Exception Status
$0014	0 - 2	SCI Receive Data
$0016	0 - 2	SCI Receive Data with Exception Status
$0018	0 - 2	SCI Transmit Data
$001A	0 - 2	SCI Idle Line
$001C	0 - 2	SCI Timer
$001E	3	NMI — Reserved for Hardware Development
$0020	0 - 2	Host Receive Data
$0022	0 - 2	Host Transmit Data
$0024	0 - 2	Host Command (Default)
$0026	0 - 2	Available for Host Command
$0028	0 - 2	Available for Host Command
$002A	0 - 2	Available for Host Command
$002C	0 - 2	Available for Host Command
$002E	0 - 2	Available for Host Command
$0030	0 - 2	Available for Host Command
$0032	0 - 2	Available for Host Command
$0034	0 - 2	Available for Host Command
$0036	0 - 2	Available for Host Command
$0038	0 - 2	Available for Host Command
$003A	0 - 2	Available for Host Command
$003C	0 - 2	Available for Host Command
$003E	3	Illegal Instruction

Tabelle 2.7: Lage der Interruptvektoren im Programmspeicher und zulässige Prioritätslevel

Anders als beim Mikroprozessor werden Mehrwortbefehle vor Eintritt in eine Interruptroutine nicht fertig abgearbeitet, sondern abgebrochen und später wieder aufgenommen. Ein in Abarbeitung befindlicher Einwortbefehl wird dagegen stets fertig bearbeitet.

Spezielle Interrupts wie NMI, TRACE und SWI sind beim ***Debuggen*** von Programmen nützlich. TRACE erzeugt z.B. einen Interrupt nach jedem Befehl, wenn das T-Flag im Status Register SR gesetzt ist (Single-Step-Betrieb). SWIs (Software-Interrupts) ermöglichen das Einstellen von ***Breakpoints***.

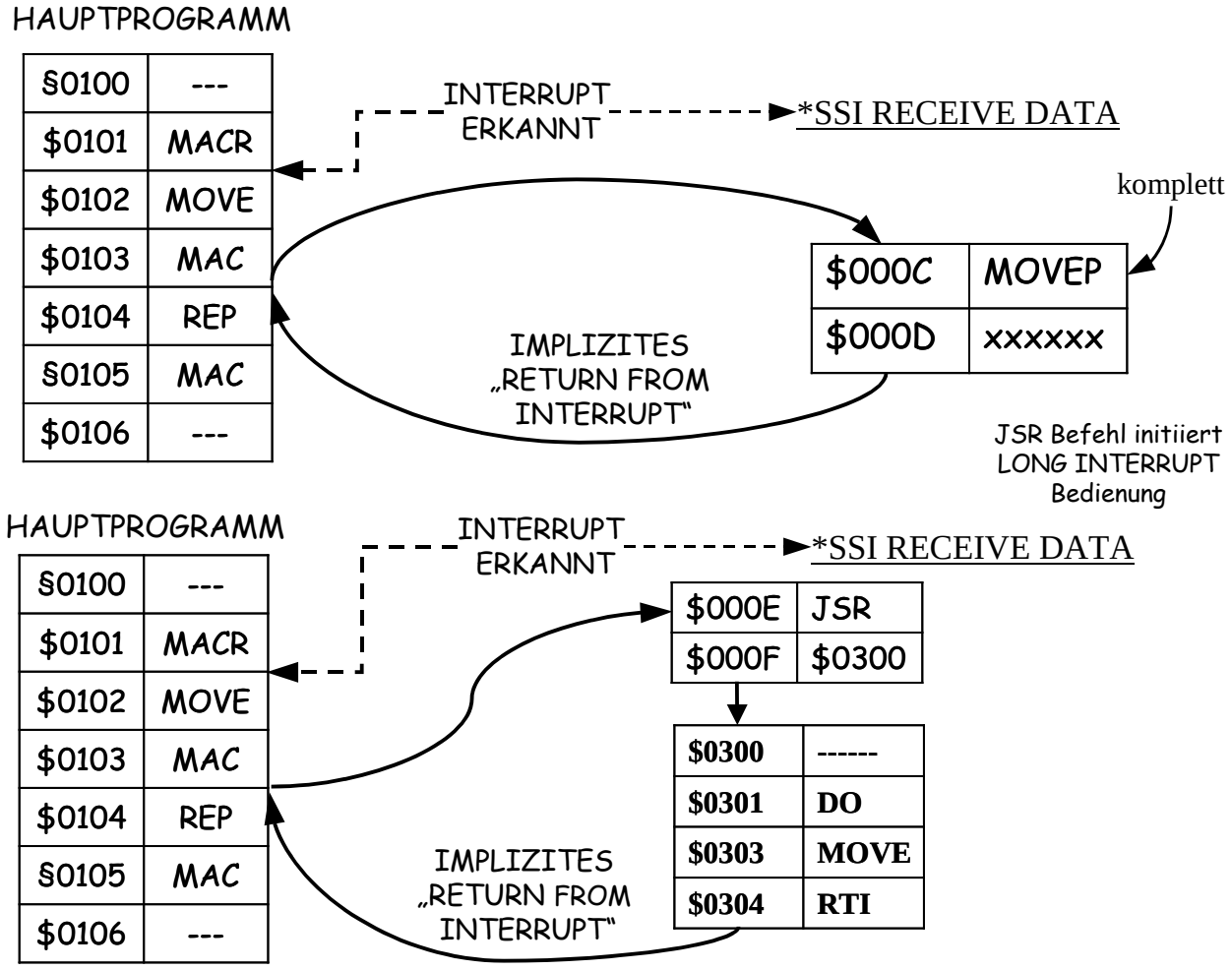


Bild 2.12: Illustration des Unterschieds zwischen ’fast’ und ’long’ Interrupt

Prioritätseinstellung im Statusregister mit den Interrupt-Maskierungsbits I0, I1

I1	I0	zugelassen	ausmaskiert
0	0	IPL 0, 1, 2 ,3	keiner
0	1	IPL 1, 2, 3	IPL 0
1	0	IPL 2, 3	IPL 0, 1
1	1	IPL 3	IPL 0, 1, 2

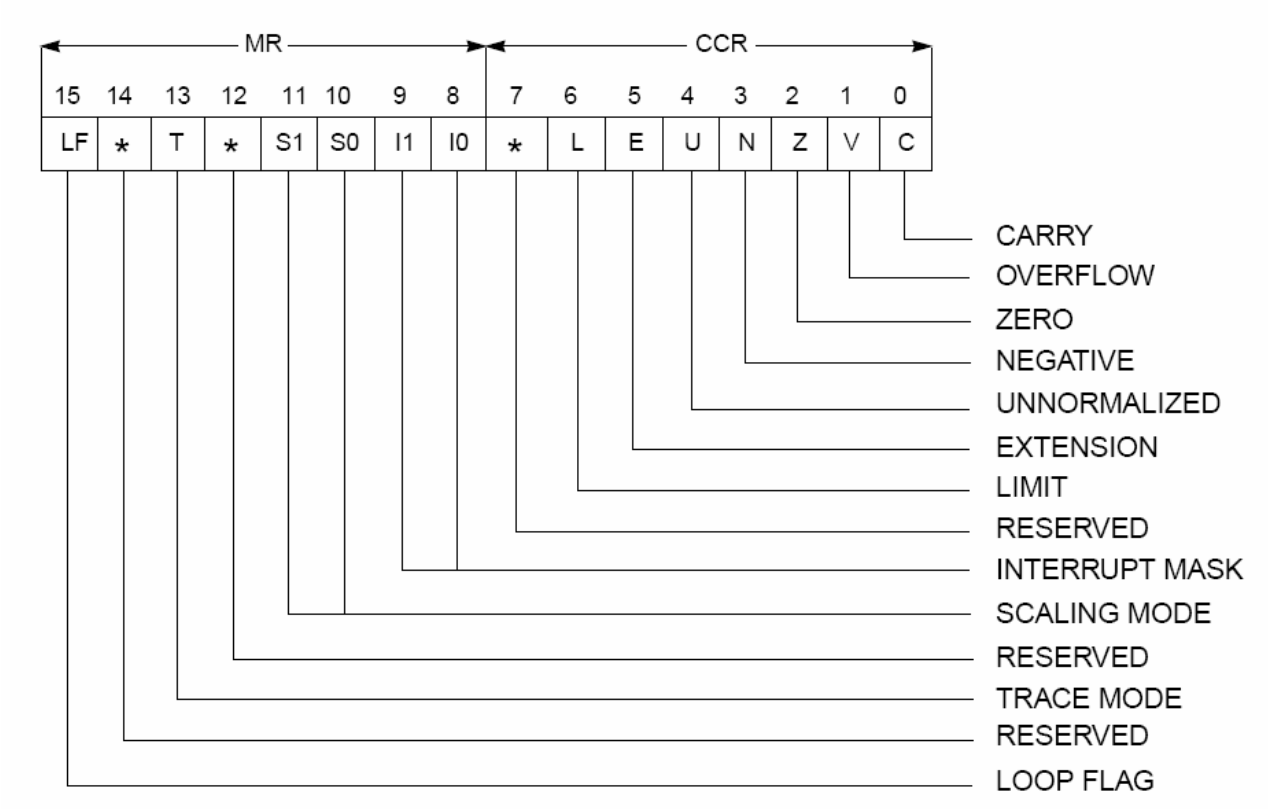


Bild 2.13: Übersicht über das gesamte Statusregister

LF	*	T	*	S1	S0	**I1**	**I0**
						⇑	⇑

Interrupt–Maskierung im **MR**

Das Interrupt–Prioritätsregister (IPR Adresse im X-Datenspeicher: X:$FFFF)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCL0	SCL0	SSL1	SSL0	HPL1	HPL0	0	0	0	0	IBL2	IBL1	IBL0	IAL2	IAL1	IAL0

Bits zur Prioritätseinstellung

xxL1	xxL0	zugelassen	Ebene
0	0	nein	–
0	1	ja	0
1	0	ja	1
1	1	ja	2

Auslösung der externen Interrupts IRQA, B	
IBL2, IAL2	ausgelöst durch
0	Pegel
1	Rückflanke

Befehlsabarbeitung in der Pipeline

⇓ Zyklus-Nummer

Aktion	1	2	3	4	5	6	7	8
Befehl holen	I	II	III_1	III_2	IV	V_1	V_2	VI
Dekodieren	-	I	II	III_1	III_2	IV	V_1	V_2
Ausführen	-	-	I	II	III_1	III_2	IV	V_1

Wann und wie wird die Pipeline für den Programmierer sichtbar und kann Probleme verursachen?

Fallstudien:

```
ORI     #04,OMR

MOVE    X:$100,A
```

OMR

.....	DE	MB	MA

⇑

DE=0 ⇒ externes RAM wird angesprochen

DE=1 ⇒ internes ROM wird angesprochen

Hier wird anstelle des gewünschten Inhalts vom internen ROM-Platz $100 der des externen RAM-Platzes $100 in den Akku geholt, falls vorher DE=0 war.

Lösung:

```
ORI     #04,OMR
NOP
MOVE    X:$100,A
```

Ein weiteres Beispiel:

```
MOVE    #25,R3
MOVE    X:(R3),A
```

Hier wird nicht der Inhalt der Stelle 25 des X-Datenspeichers in den Akku geholt, sondern der frühere Inhalt von R3, der vor dem ersten MOVE-Befehl vorhanden war, bestimmt, von welcher X-Speicherstelle Daten in den Akku gebracht werden.

Lösung:

```
MOVE #25,R3
NOP
MOVE X:(R3),A
```

Besondere Vorsicht ist bei Interrupts geboten, weil:

- 2 Zusatzzyklen zur Prioritätsprüfung ablaufen müssen
- die Interrupt-Latenz in der Pipeline 4 Befehlszyklen lang ist

Beispiele:

Hier werden die Steuerbits des Mode-Registers MR, das die oberen 8bit des Statusregisters SR einnimmt – siehe Bild 2.13.

```
        ORI    #03,MR        ;vorher sei der Inhalt von MR xxxxxx00 gewesen
                             ;d.h. alle Interrupts waren zugelassen
        Bef. 1               ;hier würde ein INT mit IPL0...2 noch angenommen
        Bef. 2               ;auch hier würde ein INT mit IPL0...2 noch angenommen
----------------------
        Bef. 3               ;⇐ Interrupt (IPL0...2) tritt auf und wird ignoriert


        ANDI   #00,MR        ;vorher sei der Inhalt von MR xxxxxx11 gewesen
        Bef. 1               ;hier würde kein INT mit IPL0...2 angenommen
        Bef. 2               ;auch hier würde kein INT mit IPL0...2 angenommen
----------------------
        Bef. 3               ;⇐ Interrupt (IPL0...2) tritt auf und wird angenommen
```

Die Pipeline bei der Bedienung aufeinander folgender FAST-Interrupts

	↯ Interrupt registriert						↯ Int. Freigabe	
1. Int. Steuerzyklus	i						i	
2. Int. Steuerzyklus		i						i
Befehl holen	B. 1	B. 2	IN-1	IN-2	B. 3	B. 4		
Dekodieren		B. 1	B. 2	IN-1	IN-2	B. 3	B. 4	
Ausführen			B. 1	B. 2	IN-1	IN-2	B. 3	B. 4
Zyklus-Nummer	1	2	3	4	5	6	7	8

Die Pipeline bei der Abarbeitung einer LONG-Interruptbedienung

	↯ Interrupt registriert							↯ erneute Interruptfreigabe							
1. STZ	i														
2. STZ		i													
Befehl holen	B.1	B.2	IN1	IN2	IN3	IN4	IN5	IN6	IN7	RTI	–	B.3	B.4		
Dekodieren		B.1	B.2	IN1	IN2	IN3	IN4	IN5	IN6	IN7	RTI	nop	B.3	B.4	
Ausführen			B.1	B.2	IN1	IN2	IN3	IN4	IN5	IN6	IN7	RTI	nop	B.3	B.4
Zyklus-Nr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Innerhalb der gleichen Prioritätsebene ist die Festlegung einer Rangfolge erforderlich, um Eindeutigkeit zu erhalten.

Priority	Exception	Enabled By	Bit No.	X Data Memory Address
Level 3 (Nonmaskable)				
Highest	Hardware $\overline{\text{RESET}}$	—	—	—
	III	—	—	—
	NMI	—	—	—
	Stack Error	—	—	—
	Trace	—	—	—
Lowest	SWI	_	—	—
Levels 0, 1, 2 (Maskable)				
Highest	$\overline{\text{IRQA}}$ (External Interrupt)	$\overline{\text{IRQA}}$ Mode Bits	0 and 1	$FFFF
	$\overline{\text{IRQB}}$ (External Interrupt)	$\overline{\text{IRQB}}$ Mode Bits	3 and 4	$FFFF
	Host Command Interrupt	HCIE	2	$FFE8
	Host Receive Data Interrupt	HRIE	0	$FFE8
	Host Transmit Data Interrupt	HTIE	1	$FFE8
	SSI RX Data with Exception Interrupt	RIE	15	$FFED
	SSI RX Data Interrupt	RIE	15	$FFED
	SSI TX Data with Exception Interrupt	TIE	14	$FFED
	SSI TX Data Interrupt	TIE	14	$FFED
	SCI RX Data with Exception Interrupt	RIE	11	$FFF0
	SCI RX Data Interrupt	RIE	11	$FFF0
	SCI TX Data Interrupt	TIE	12	$FFF0
	SCI Idle Line Interrupt	ILIE	10	$FFF0
Lowest	SCI Timer Interrupt	TMIE	13	$FFF0

Tabelle 2.8: Das im DSP5600x implementierte Pollingschema

Beispiel für eine Interrupt-Verschachtelung

Ein Softwareinterrupt (SWI) verhindert die Abarbeitung eines Low-Level-Interrupts, der vorher aufgetreten war und angenommen wurde.

1. STZ		i		i*							
2. STZ			i		i*						
Befehl holen	B.3	B.4*	B.5	IN1	–	I1_S	I2_S	I1_U	I2_U	I3_U	I4_U
Dekodieren	B.2	B.3	SWI	–	–	–	JSR	–	I1_U	I2_U	I3_U
Ausführen	B.1	B.2	B.3	SWI	NOP	NOP	NOP	JSR	–	I1_U	I2_U
Zyklus-Nr.	1	2	3	4	5	6	7	8	9	10	11

B.4* ist ein Softwareinterrupt, der erst beim Dekodieren in der Pipeline erkannt wird

i normale Interruptanforderung, Level 0...2

i* Interruptanforderung durch SWI, Level 3

Fast Interrupts sind grundsätzlich nicht unterbrechbar. Wenn jedoch innerhalb der Steuerzyklen ein Interrupt höherer Priorität auftritt, kann die Abarbeitung selbst nach einem OP-Code-Holen von der 1. Interrupt-Vektoradresse gestoppt werden, wie obiges Beispiel zeigt.

2.2 Timersysteme und Pulsweitenmodulation

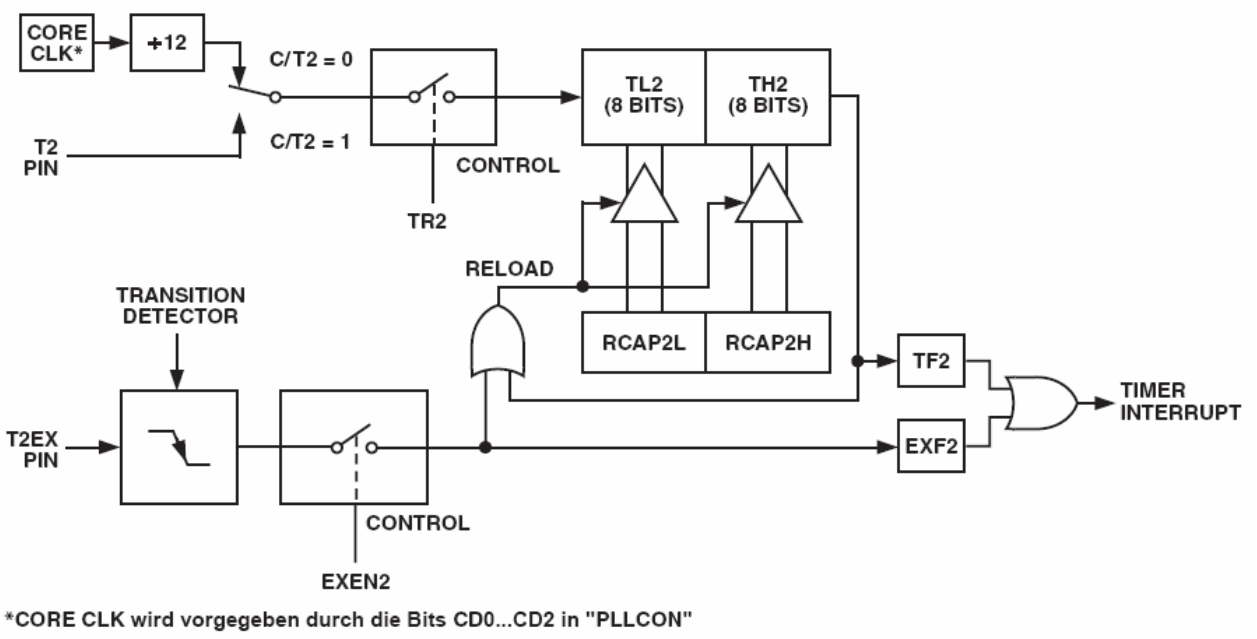


Bild 2.14: Einfaches Timersystem in einem 8bit-Mikrocontroller (8052)

Tabelle 2.9: Steuerregister (T2CON) für das Timersystem nach Bild 2.14 und Bild 2.15

Bit-Nr.	Name	Beschreibung
7	TF2	Timer 2 Overflow Flag Set by hardware on a Timer 2 overflow. TF2 will not be set when either RCLK = 1 or TCLK = 1. Cleared by user software.
6	EXF2	Timer 2 External Flag Set by hardware when either a capture or reload is caused by a negative transition on T2EX and EXEN2 = 1. Cleared by user software.
5	RCLK	Receive Clock Enable Bit Set by the user to enable the serial port to use Timer 2 overflow pulses for its receive clock in serial port Modes 1 and 3. Cleared by the user to enable Timer 1 overflow to be used for the receive clock.
4	TCLK	Transmit Clock Enable Bit Set by the user to enable the serial port to use Timer 2 overflow pulses for its transmit clock in serial port Modes 1 and 3. Cleared by the user to enable Timer 1 overflow to be used for the transmit clock.
3	EXEN2	Timer 2 External Enable Flag Set by the user to enable a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. Cleared by the user for Timer 2 to ignore events at T2EX.
2	TR2	Timer 2 Start/Stop Control Bit Set by the user to start Timer 2. Cleared by the user to stop Timer 2.
1	CNT2	Timer 2 Timer or Counter Function Select Bit Set by the user to select counter function (input from external T2 pin). Cleared by the user to select timer function (input from core clock).
0	CAP2	Timer 2 Capture/Reload Select Bit Set by the user to enable captures on negative transitions at T2EX if EXEN2 = 1. Cleared by the user to enable autoreloads with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the timer is forced to autoreload on Timer 2 overflow.

In einfachen 8bit-Mikrocontrollern werden Timer meist zur Erzeugung von Zeitmarken, zum „Vermessen“ von Impulsdauern oder zur Baudratengenerierung verwendet. In vielen Fällen benötigt man eine Wortlänge, die über 8bit hinausgeht. Dazu werden in der Regel zwei Timerregister verknüpft, so daß man 16bit zur Verfügung hat – vgl. Bild 2.14. In erweiterten 8 bit-Mikrocontrollern wie z.B. dem 80C537 (Siemens) oder der Mikroconverterfamilie ADuC8xx (Analog Devices) werden Timer für die Pulsweitenmodulation eingesetzt. Hierbei ist meist eine deutliche Erhöhung der Taktfrequenz erforderlich.

2.2.1.1 Baudratenerzeugung

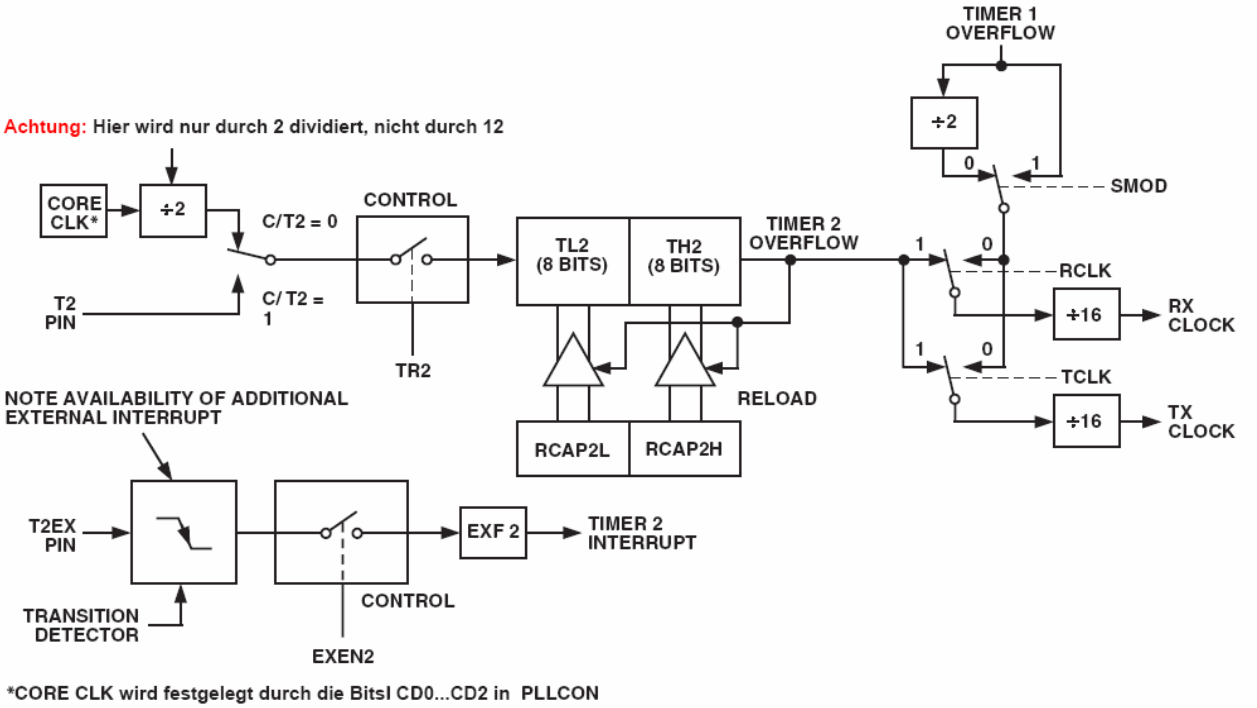


Bild 2.15: Timer 2 zur Baudratenerzeugung. Zu beachten ist hier, daß der Prozessortakt dem Timer nur durch 2 und nicht durch 12 geteilt zugeführt wird.

Tabelle 2.10: Beispiele zur Baudratengenerierung im ADuC 832 mittels Timer 2

einige gebräuchliche mit Timer 2 erzeugbare Baudraten (ADuC832)					
Idealwert der Baudrate	Kerntakt (MHz)	**RCAP2H** HEX	**RCAP2L** HEX	realer Wert der Baudrate	**Fehler in %**
19200	16,777216	FF	E5	19418	+1,135
9600	16,777216	FF	C9	9532,51	–0,703
4800	16,777216	FF	93	4809,98	+0,208
2400	16,777216	FF	26	2404,99	+0,0208
1200	16,777216	FE	4B	1199,74	–0,0213
9600	2,097152	FF	F9	9362,29	–2,476
4800	2,097152	FF	F2	2681,14	–2,476
2400	2,097152	FF	E5	2427,26	+1,135
1200	2,097152	FF	C9	1191,56	–0,703

Der ADuC 832 verwendet eine PLL[4] zur Vervielfachung der Frequenz des angeschlossenen Quarzes. Typischerweise wird ein „Uhrenquarz“ mit der Frequenz f_u=32,768kHz = 2^{15} Hz benutzt.

Wie man sieht, läßt sich aus der Uhrenquarzfrequenz der in der Regel für eine Uhr benötigte Sekundentakt einfach mit einem 15-stufigen Binärteiler gewinnen.

[4] Phase Locked Loop

Die PLL erzeugt eine mit dem Faktor 512 multiplizierte Ausgangsfrequenz, d.h. f_c=32,768·512= 16.777.216 Hz ≈ 16,8MHz, die zur Speisung des Rechnerkerns mit einem programmierbaren 7–stufigen Binärteiler an die jeweiligen Aufgaben angepaßt, d.h. durch 2^0...2^7 geteilt werden kann. Dazu sind die 3 bit (CD0...CD3) im SRF ***PLLCON*** entsprechend einzustellen.

Nach Bild 2.15 berechnet sich die Baudrate dann zu

$$b = \frac{f_c}{32 \cdot (2^{16} - RCAP2H, RCAP2L)} \qquad (2.2)$$

Über Timer 0...Timer 2 hinaus verfügt der ADuC832 über einen weiteren Timer 3 –s. Bild 2.16, der speziell für die Baudratengenerierung vorgesehen ist.

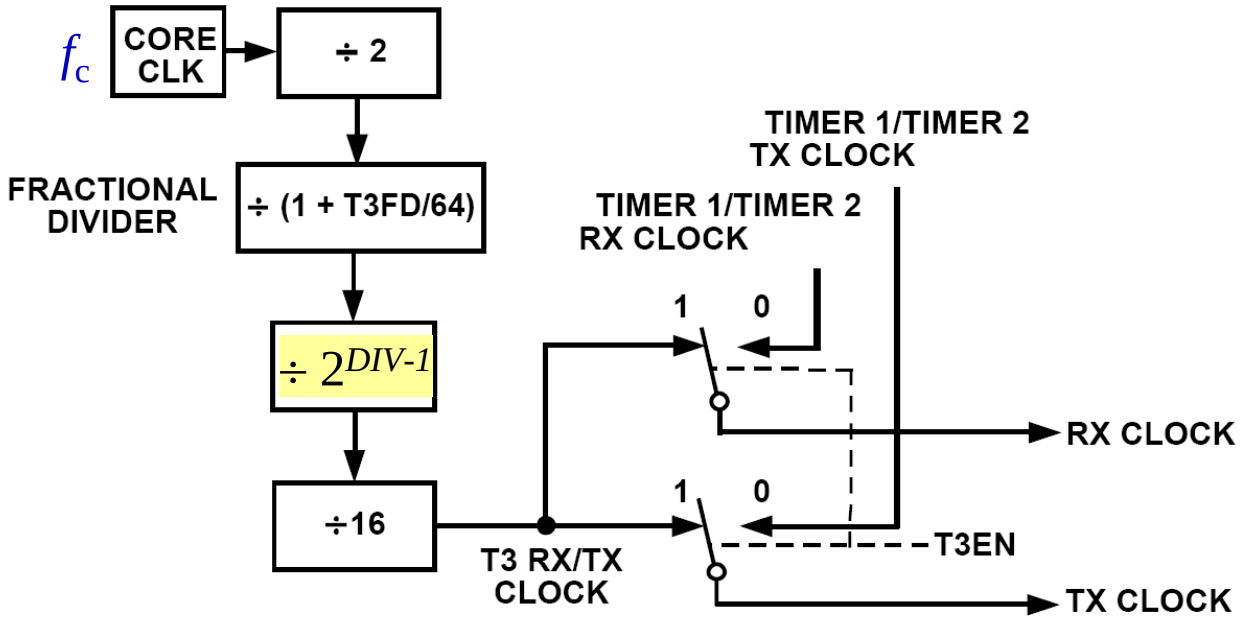


Bild 2.16: Ein spezieller Timer zur flexibeln Baudratenerzeugung (T3 im ADuC832)

Nachdem der Kerntakt durch 2 geteilt ist, gelangt er auf einen programmierbaren „Fraktionalteiler“, der eine weitere Teilung durch (1+T3FD/64) vornimmt, wobei T3FD ein SFR ist, das mit einer Zahl zwischen 0 und 255 beschrieben werden kann. Es folgt ein weiterer Teiler (2^{DIV-1}), der nur durch Zweierpotenzen im Bereich 2^0...2^7 teilen kann. Die entsprechenden Bits DIV0...DIV2 sind im SFR T3CON passend einzustellen – s. Tabelle 2.11.

Das Setzen von Bit 7 T3BAUDEN wählt Timer 3 als Baudratengenerator aus und stellt damit die Timer 1 und 2 von dieser Aufgabe frei, so daß sie für andere Zwecke zur Verfügung stehen.

Wozu ein weiterer Timer 3?

Die gewöhnlichen Frequenzteiler in Standard UARTs arbeiten mit Ganzzahlen und sind daher oft nicht in der Lage, alle gewünschten Baudraten z.B. mit einem 12 MHz-Quarz zu erzeugen. Beim Standard 8052-Mikrocontroller lassen sich so keine höheren Standardbaudraten als 4800 generieren.

Dieses Problem löst Timer 3 im ADuC832. Dieser Timer ist speziell für das Erzeugen hoher Baudraten, z.B. bis 230.400 ausgelegt. Darüber hinaus erlaubt er eine sehr feine Stufung, d.h. von 12 bit/s bis 393.216 bit/s kann jede Rate mit einem Fehler von maximal ±0,8% generiert werden.

Tabelle 2.11: Aufbau des SFR T3CON im ADuC832

Bit	Name	Beschreibung			
7	T3BAUDEN	T3 UART BAUD ENABLE Bit: Selektiert Timer 3 als Baudratengenerator. Wenn es gesetzt wird, werden die Bits PCON.7, T2CON.4 and T2CON.5 ignoriert. Wenn es “0” ist, erfolgt die Baudratengenerierung wie in einem Standard-8052-Mikrocontroller			
6		–			
5		–			
4		–			
3		–			
2	DIV2	binärer Teilfaktor			
1	DIV1	**DIV2**	**DIV1**	**DIV0**	**Teilfaktor**
0	DIV0	0	0	0	1
		0	0	1	2
		0	1	0	4
		0	1	1	8
		1	0	0	16
		1	0	1	32
		1	1	0	64
		1	1	1	128

Wie aus Bild 2.16 und Tabelle 2.11 hervorgeht, wird Timer 3 mit den beiden Spezialfunktionsregistern T3CON und T3FD eingestellt. In T3CON wird u.a. der Binärteiler DIV programmiert, der im Zusammenspiel mit dem ’fraktionalen Teiler’ die passende Frequenz an den 16-er Teiler der Schnittstelle liefert. Mit der Kerntaktfrequenz f_c ergibt sich dann aus Bild 2.16 die Baudrate

$$b = \frac{f_c}{32 \cdot 2^{DIV-1}\left[1+\frac{T3FD}{64}\right]} = \frac{2 f_c}{2^{DIV-1}\cdot\left(T3FD+64\right)}. \qquad (2.3)$$

Die folgende Tabelle 2.12 zeigt eine Liste gebräuchlicher Baudraten im Bereich 9600 bis 230.400 und die bei Timer 3 benötigten Einstellungen zu ihrer Erzeugung. Wie man sieht, bleibt der Fehler stets deutlich unter 1%, so daß aufgrund der Numerik keine Degradation zu erwarten ist.

Tabelle 2.12: Baudratenerzeugung mit dem Timer 3 im ADuC832

ideale Baud-rate	CD	f_c MHz	DIV	2^{DIV}	T3CON	T3FD	reale Baudrate	**Fehler in %**
230.400	0	16,77	2	2	82H	09H	229.825	–0,25
115200	0	16,77	3	4	83H	09H	114.912	–0,25
115200	1	8,388	2	2	82H	09H	114.912	–0,25
115200	2	4,194	1	1	81H	09H	114.912	–0,25
57600	0	16,77	4	8	84H	09H	57.456	–0,25
57600	1	8,388	3	4	83H	09H	57.456	–0,25
57600	2	4,194	2	2	82H	09H	57.456	–0,25
57600	3	2,097	1	1	81H	09H	57.456	–0,25
38400	0	16,77	4	8	84H	2DH	38.480	0,208
38400	1	8,388	3	4	83H	2DH	38.480	0,208
38400	2	4,194	2	2	82H	2DH	38.480	0,208
38400	3	2,097	1	1	81H	2DH	38.480	0,208
19200	0	16,77	5	16	85H	2DH	19.240	0,208
19200	1	8,388	4	8	84H	2DH	19.240	0,208
19200	2	4,194	3	4	83H	2DH	19.240	0,208
19200	3	2,097	2	2	82H	2DH	19.240	0,208
19200	4	1,048	1	1	81H	2DH	19.240	0,208
9600	0	16,77	6	32	86H	2DH	9.620	0,208
9600	1	8,388	5	16	85H	2DH	9.620	0,208
9600	2	4,194	4	8	84H	2DH	9.620	0,208
9600	3	2,097	3	4	83H	2DH	9.620	0,208
9600	4	1,048	2	2	82H	2DH	9.620	0,208
9600	5	0,524	1	1	81H	2DH	9.620	0,208

2.2.1.2 Pulsweitenmodulation

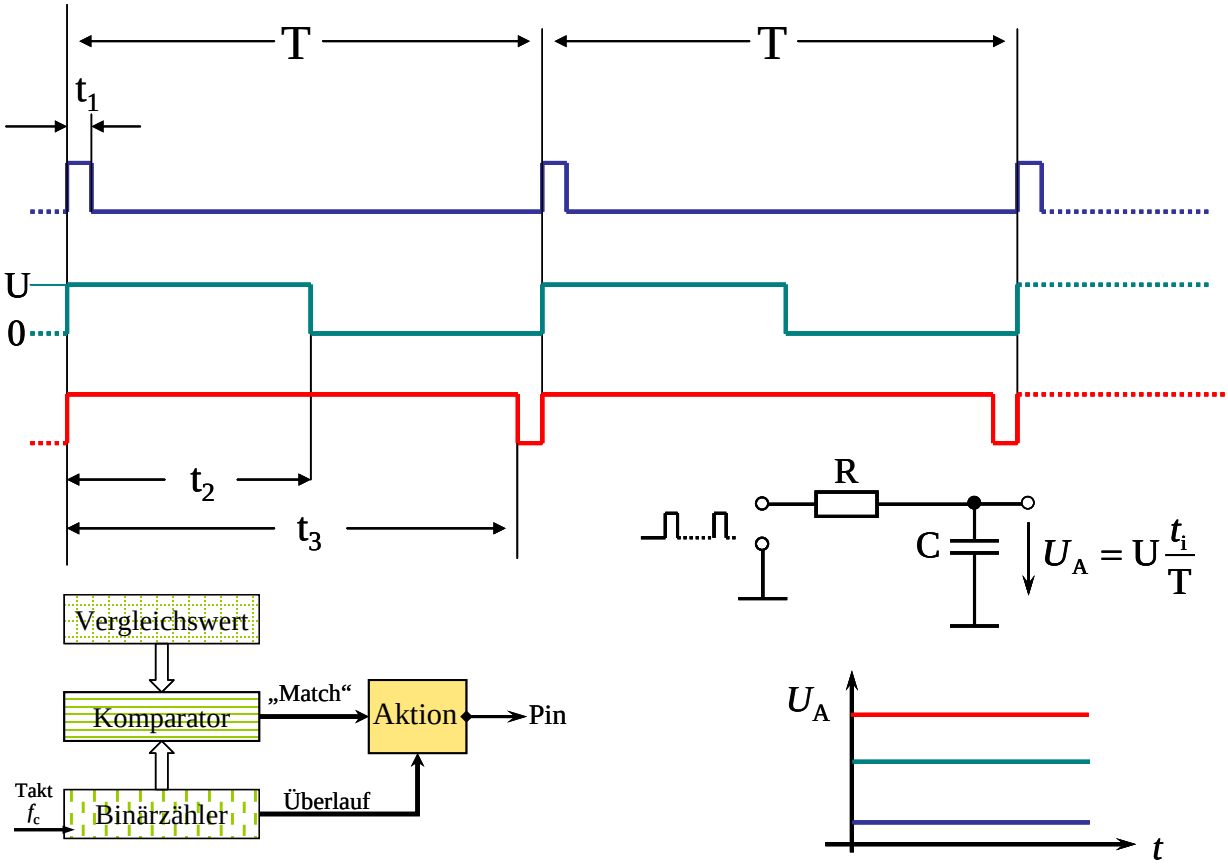


Bild 2.17: Prinzip der Pulsweitenmodulation und zugehörige Hardware

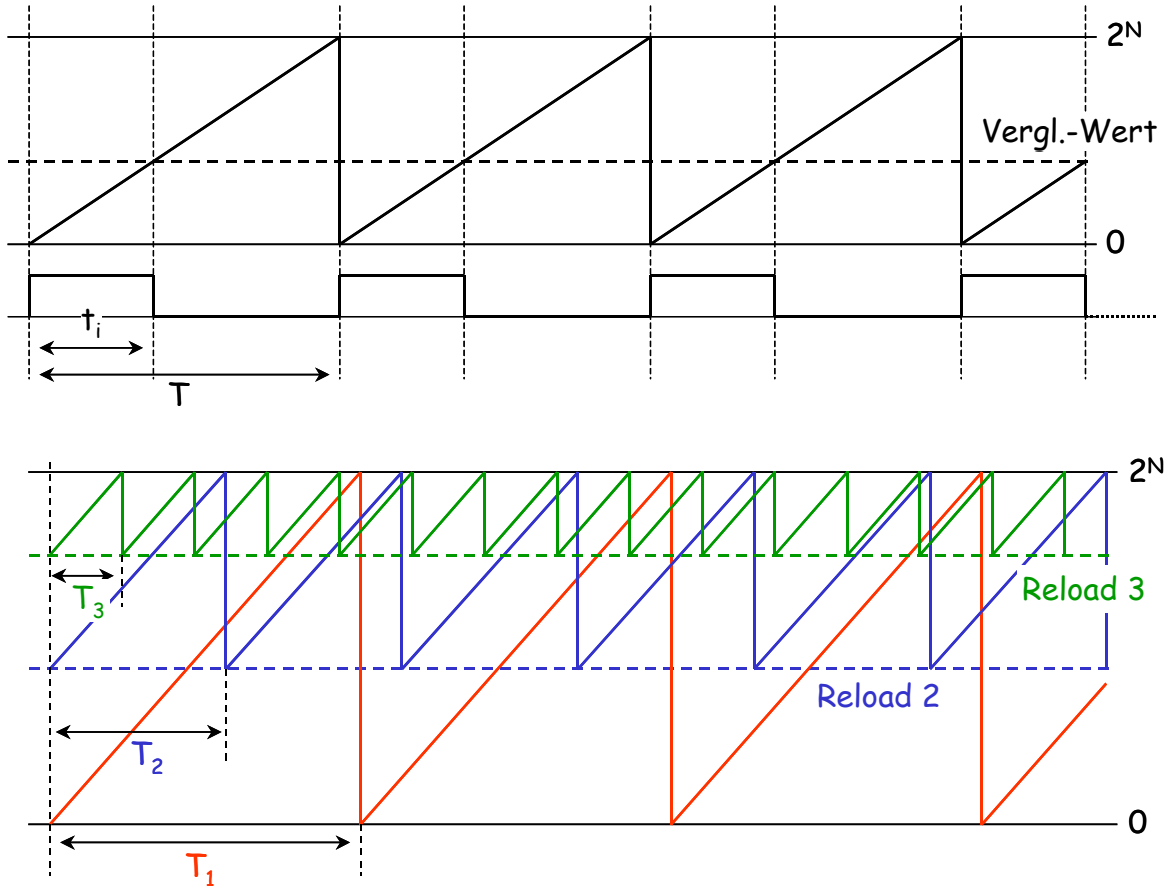


Bild 2.18: Grundlegende Timerfunktionen für die Pulsweitenmodulation

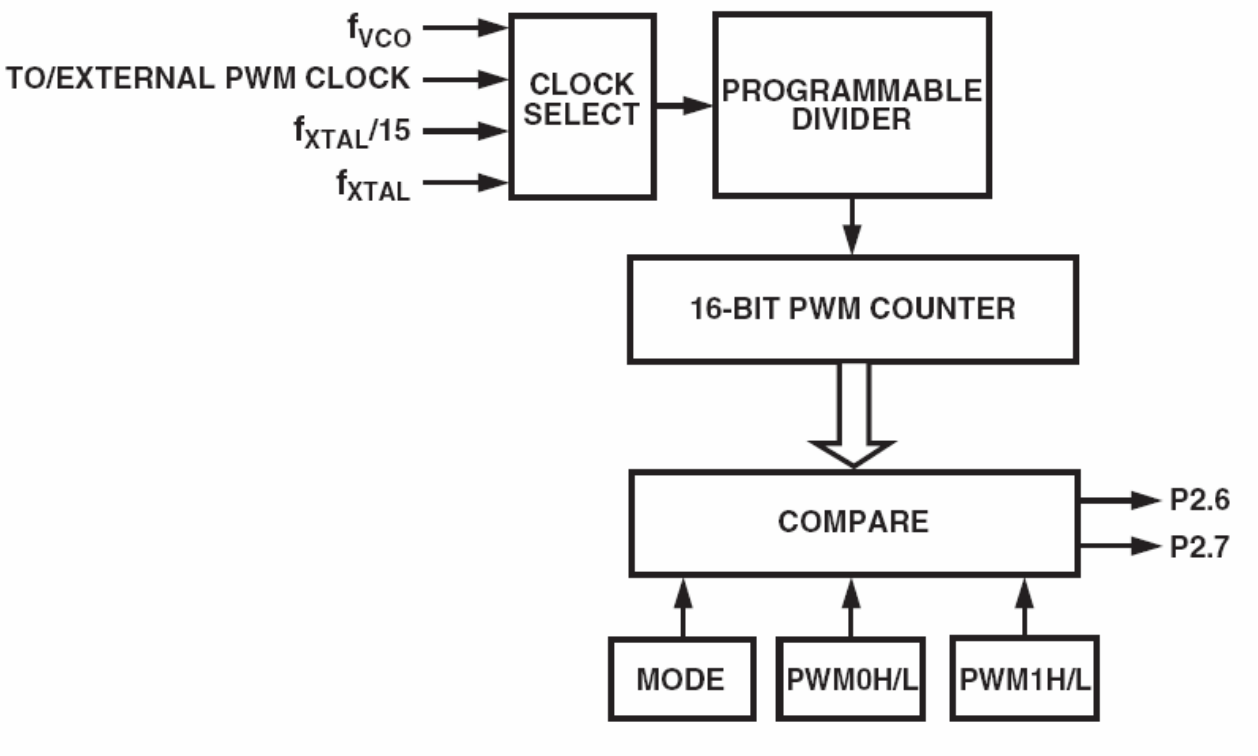
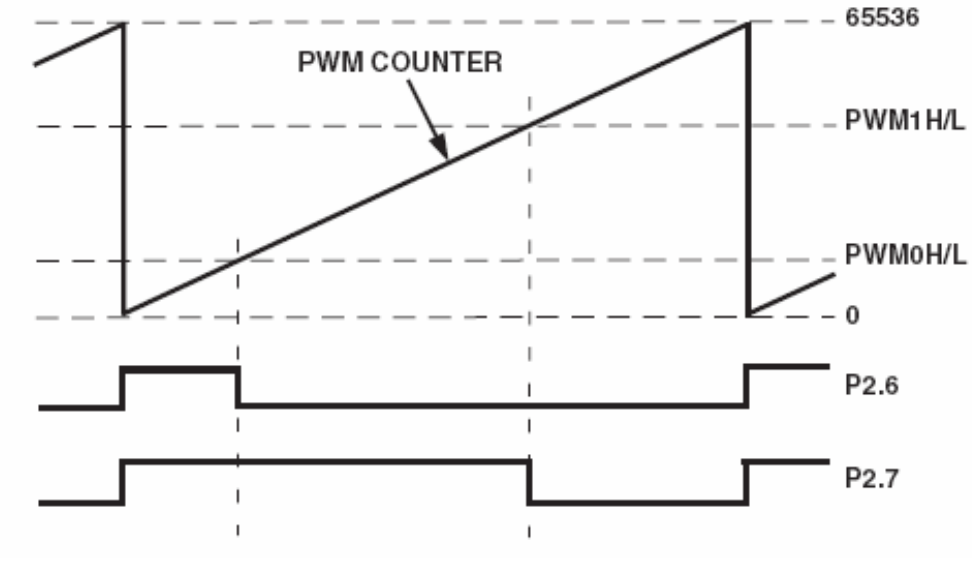


Bild 2.19: PWM-Realisierung im Mikroconverter ADuC832

Im folgenden werden beispielhaft zwei verschiedene PWM-Betriebsarten (MODES) untersucht.

Bild 2.20 zeigt eine PWM-Betriebsart, bei der der Zähler stets von 0 bis 65536 aufwärts zählt und dann wieder mit 0 beginnt. Wenn der Basistakt f_{VCO}=16,777 MHz=2^{24} Hz gewählt wird, erhält man eine 'PWM-Ausgangsrate' von 256 Hz ($2^{24}/2^{16}=2^8$).



Das Tastverhältnis an den Ausgangspins P2.6 und P2.7 kann individuell programmiert werden. Wie Bild 2.20 zeigt, wird bei jedem Zählerüberlauf eine Vorderflanke an den Ausgangspins erzeugt. Solange der Zählerstand kleiner als PWM0H/L ist, bleibt P2.6 auf „1".

Bild 2.20: Zweifach-PWM mit 16 bit Auflösung

Wenn der Zählerstand den Wert PWM0H/L erreicht, geht P2.6 auf Null und verbleibt dort, bis ein weiterer Zählerüberlauf auftritt. Entsprechend verhält sich P2.7 im Zusammenspiel mit PWM1H/L.

Die beiden PWM-Ausgänge sind also synchronisiert, d.h. bei jedem Zählerüberlauf beginnt ein neuer Zyklus mit einer Vorderflanke.

Wie schon aus Bild 2.17 hervorgeht, kann die PWM in einfacher Weise zur D/A-Wandlung verwendet werden. Die Möglichkeiten sind um so vielfältiger, je schneller eine solche Wandelfunktion ist. Hierbei ist die Periodendauer T die entscheidende Größe, weil der Rekonstruktionstiefpaß nach ihr auszulegen ist – vgl. Bild 2.17.

Die zweite PWM-Betriebsart, die hier untersucht wird, spiegelt das Prinzip der Sigma-Delta-Wandlung wieder, die später noch im Detail betrachtet wird. Es wird im weiteren davon ausgegangen, daß die PWM-Taktfrequenz 16,777216 MHz (2^{24}Hz) beträgt. Das folgende Bild 2.21 zeigt eine Blockschaltung für diese spezielle Betriebsart.

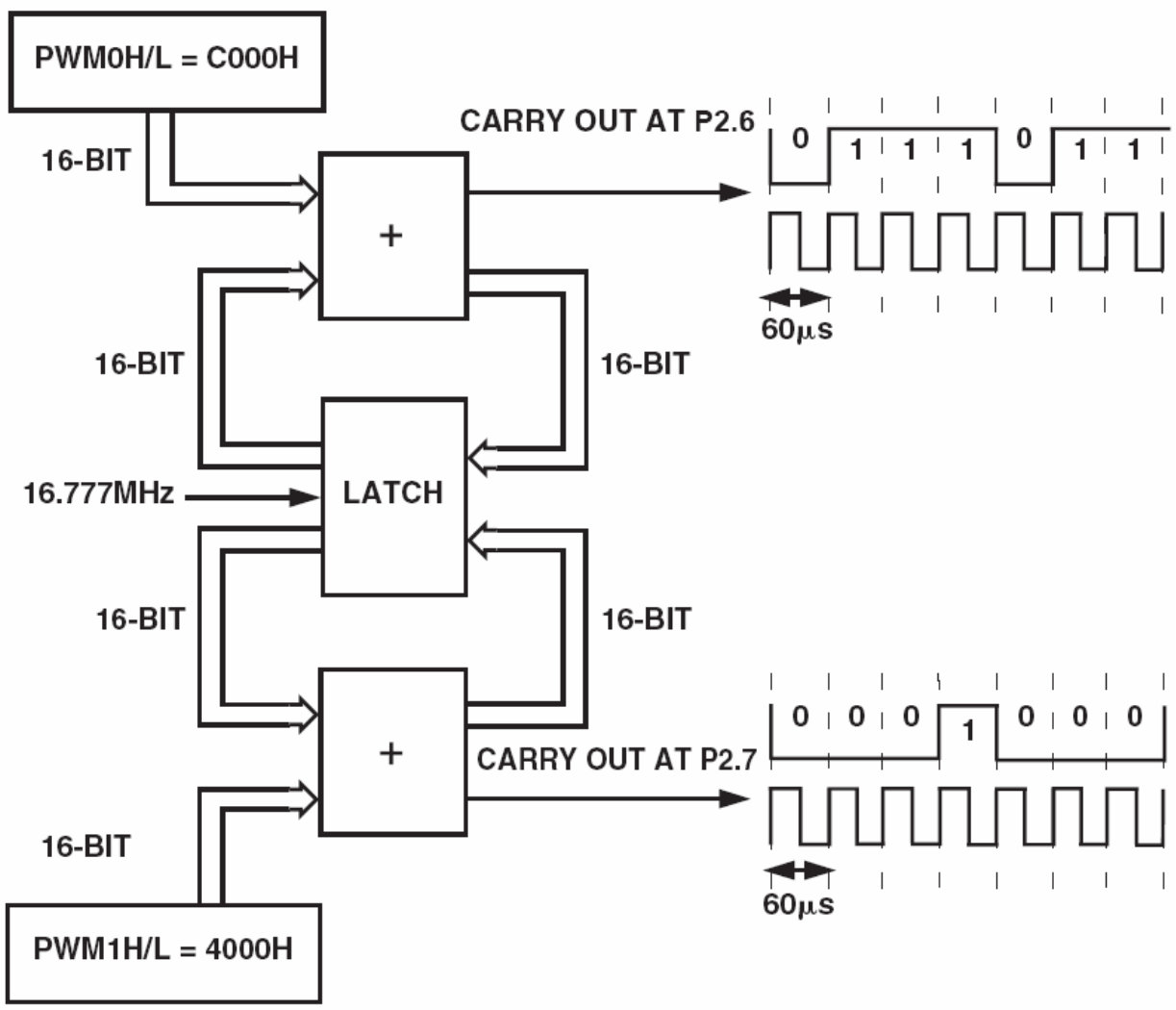


Bild 2.21: PWM als ’Sigma-Delta’-D/A-Wandler

Wie man sieht, können wieder zwei „Kanäle“ unabhängig voneinander parallel betrieben werden.

Die Kernstücke des PWM als Sigma-Delta-D/A-Wandler sind ein Addierer und ein Zwischenspeicher (Latch). In die SFRs PWM0H/L bzw. PWM1H/L werden jeweils „Startwerte“ eingetragen, die dann mit Maximalfrequenz von 2^{24} Hz in den zugehörigen Latches akkumuliert werden. Die Besonderheit ist hier aber, daß das Latch auf 16 bit begrenzt ist und die Zählerüberläufe nicht bei der Addition berücksichtigt werden, sondern die Ausgangssignale an den Portpins 2.6 und 2.7 darstellen. Man erhält hier somit schnelle serielle Datenströme (im 60 ns-Takt) mit 1 bit Auflösung, wobei der Mittelwert nach passender Tiefpaßfilterung das gesuchte D/A-gewandelte Ausgangssignal darstellt.

Wenn man eine Periode von 2^{16}=65.536 Taktzyklen betrachtet, ist P2.6 auf „1“ für eine Zahl von Zyklen, die genau dem Inhalt der SFRs PWM0H/L entspricht und auf „0“ für (65.536–PWM0H/L) Zyklen. Das Gleiche gilt für P2.7 und PWM1H/L.

Die Zahlenbeispiele in Bild 2.21 führen mit PWM0H/L= C000H bzw. PWM1H/L=4000H auf die jeweils rechts dargestellten Impulsfolgen an den Pins 2.6 und 2.7. Bei fortlaufender Addition von 4000H ergibt sich so z.B. in der unteren Bildhälfte die Zahlenfolge 4000H, 8000H, C000H und 10000H => 0000H mit Carry = P2.7 = „1“ für eine Taktdauer. In der oberen Bildhälfte erhält man das „Komplement“ dazu weil PWM0H/L= C000H ist. Würde man PWM1H/L=4001H machen, dann ergäbe sich über 2^{16} Additionen genau ein Überlauf mehr als das reguläre Muster 00010001... vorgibt. Die führt zu einer geringen Anhebung des Mittelwertes, der die D/A-gewandelte Ausgangsspannung an Pin 2.7 wiedergibt. Die Auflösung ist 16 bit, wobei jedoch eine recht lange Periodendauer, nämlich 1/256Hz ≈ 3,9ms vorliegt.

Gemäß dem Sigma-Delta-Prinzip kann die Wandelgeschwindigkeit bei gleichzeitig verringerter Auflösung erhöht werden, indem man eine entsprechende Zahl von LSBs immer auf Null läßt. Für 12 bit Auflösung bleiben also die untersten 4 bit immer Null, d.h. die kleinstmögliche Inkrementierung des Latch-Inhalts erfolgt in Schritten von 2^4=16. Daraus folgt, das die Periodendauer um den Faktor 16 sinkt bzw. die Wandelrate um diesen Faktor auf 4096 s^{-1} wächst. Werden nur 8 bit Auflösung benötigt, dann bleiben die unteren 8 bit stets Null und die Wandelrate wächst auf 256^2 = 65.536s^{-1}, d.h. die Periodendauer ist jetzt rund 15µs.

2.2.1.3 Herausragende Merkmale der PWM-Struktur im 80C166

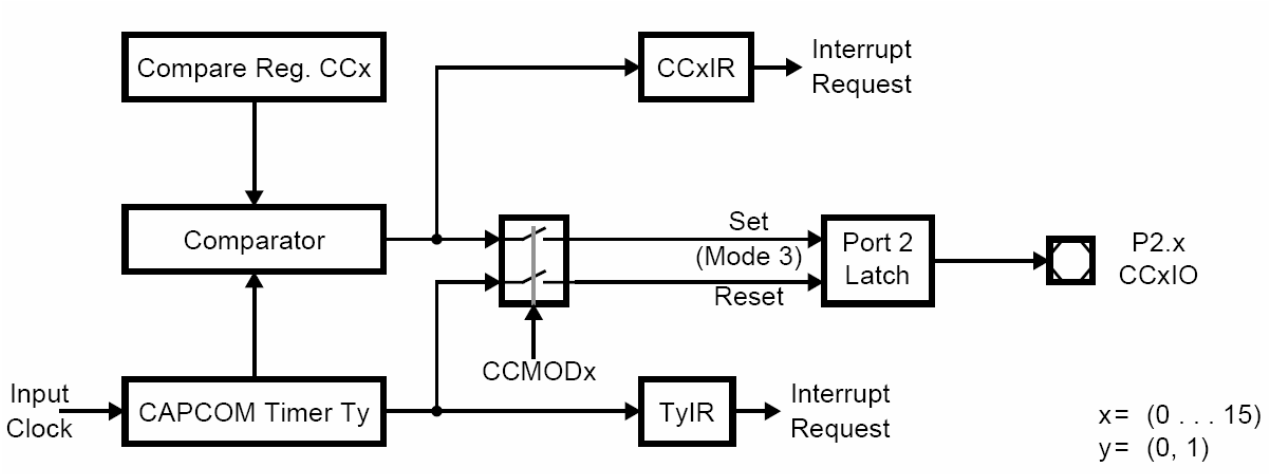


Bild 2.22: Die Compare-Modi (MODE 2 und 3) im 80C166

Wir betrachten hier exemplarisch MODE 3, bei dem im Falle eines Überlaufes des gewählten Timers (CAPCOM Timer Ty) ein Reset des zugeordneten Port-Latches erfolgt, so daß die entstehende Impulsfolge am zugehörigen Pin P2.x stets fest auf die Timerüberläufe synchronisiert ist.

Bei einem ’Compare-Match’, d.h. wenn Timerstand und Inhalt von Compare-Register CCx übereinstimmen wird dann der zugehörige Pin P2.x auf „1“ gesetzt.

Das folgende Timing- und Impulsdiagramm zeigt ein Beispiel, bei dem nach dem ersten Compare-Match der Inhalt des Registers CCx von cv1 auf cv2 geändert wurde. Man kann auf diese Weise das Tastverhältnis der Ausgangsfolge dynamisch beeinflussen.

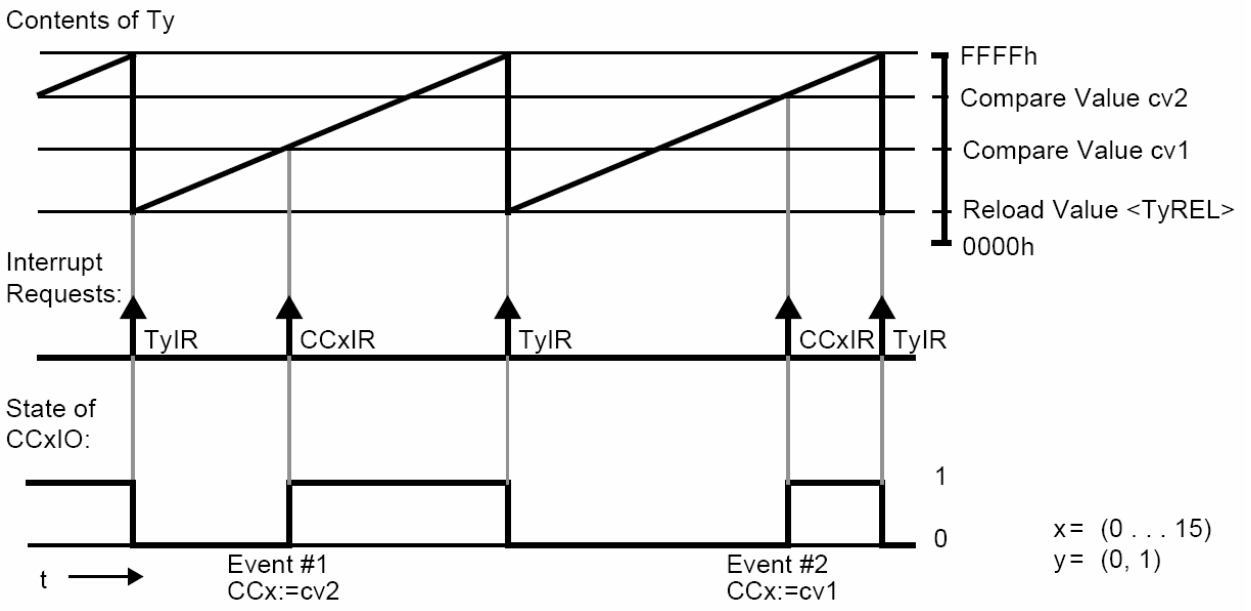


Bild 2.23: Beispiel für den Compare-MODE 3 im 80C166

Über den beschriebenen Standard-Modus hinaus bietet der 80C166 mit dem ***Double-Register-Compare-Mode*** (DRCM) eine Besonderheit, die in der modernen Antriebstechnik entscheidende Vorteile bringt. Im DRCM können losgelöst von Timerüberläufen bis zu 8 Impulsfolgen mit unterschiedlichem Tastverhältnis programmiert werden, die ohne Softwareeingriffe ablaufen.

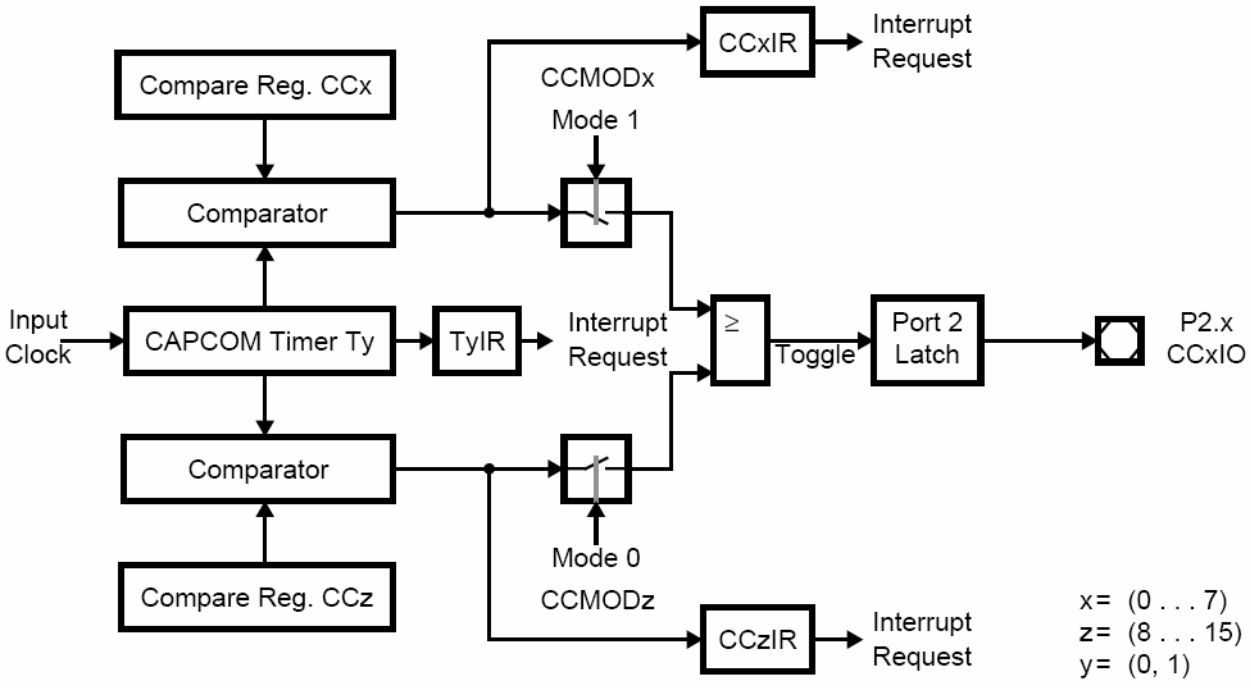


Bild 2.24: Blockschaltung zum Double-Register-Compare-Modus

Wie Bild 2.24 zeigt, ist beim DRCM immer ein Registerpaar CCx und CCz (x=0...7 und z=7..15) involviert, sowie zwei Komparatoren, die die Registerinhalte mit dem Stand von CAPCOM-Timer Ty vergleichen. Man sagt auch, daß CCx einer „Register-Bank 1" und CCz einer 'Register-Bank 2' zugeordnet sind. Jedem Register-Paar ist darüber hinaus ein Pin P2.x von Port 2 - auch mit CCxIO bezeichnet - zugeordnet, wo die programmierte Impulsfolge ausgegeben wird – vgl. Tabelle 2.13. Zur Initialisierung des DRCM muß jeweils ein Register aus Bank 1 (CC0 ... CC7) in

den Standard-Compare-Modus 1 und das zugehörige Register aus Bank 2 (CC8 ... CC15) in den Standard-Compare-Modus 0 versetzt werden.

Tabelle 2.13: Register- und Pinkonfiguration für DRCM

Bank 1	Bank2	Ausgang	Pin
CC0	CC8	CC0IO	P2.0
CC1	CC9	CC1IO	P2.1
CC2	CC10	CC2IO	P2.2
CC3	CC11	CC3IO	P2.3
CC4	CC12	CC4IO	P2.4
CC5	CC13	CC4IO	P2.5
CC6	CC14	CC6IO	P2.6
CC7	CC15	CC7IO	P2.7

Wie aus Bild 2.24 hervorgeht wird bei jeder Übereinstimmung zwischen Timer **Ty** und einem Register CCx bzw. CCz ein Kippvorgang (Toggle) im zugeordneten Port-Latch ausgelöst. Falls die Inhalte von CCx und CCz exakt gleich sind, erfolgt nur ein Kippvorgang.

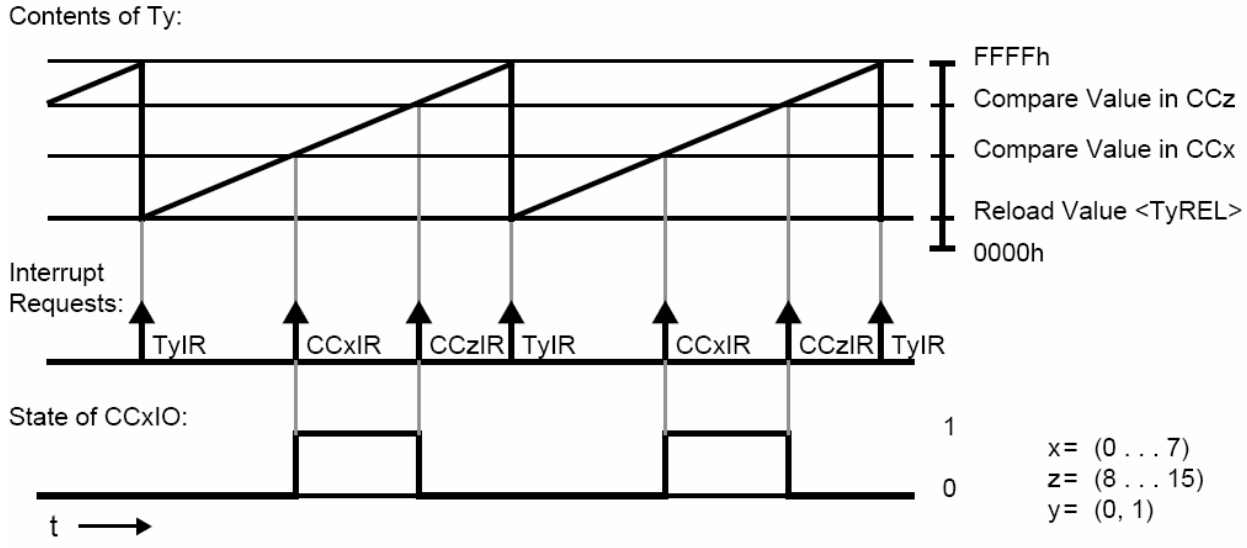


Bild 2.25: Timing- und Impulsdiagramm für den DRCM

Bild 2.25 zeigt ein Beispiel, bei dem die Inhalte von CCx und CCz konstant bleiben. Man erhält dann eine Impulsfolge mit festem Tastverhältnis am zugeordneten Pin P2.x. Die übrigen 7 Pins von Port 2 können gleichzeitig entsprechende Impulsmuster mit individuell einstellbaren Tastverhältnissen ausgeben. Eine dynamische Verstellung ist jederzeit durch Neubeschreiben der Register zwischen den Compare-Matches möglich. Dadurch, daß sowohl bei jedem Compare-Match als auch bei Timerüberlauf Interrupts generiert werden, können mögliche Kollisionen beim Neubeschreiben der Register sicher verhindert werden.

3 Digitalisierung analoger Signale

3.1 Entropie von Nachrichtenquellen

Die Entropie H ist der mittlere Informationsgehalt einer Nachrichtenquelle. H ist empirisch so definiert worden, daß eine binäre Quelle (mit zwei Zeichen) H_B=1bit hat, wenn die Zeichen mit gleicher Wahrscheinlichkeit $p(x_{1,2})$=0,5 auftreten.

Bei einer Quelle mit einem Zeichenvorrat x_i (i = 1...N) sei die Auftrittswahrscheinlichkeit eines Zeichens $p(x_i)$

- Die Erfahrung lehrt, daß der „Informationswert“ eines Zeichens x_i um so höher ist, je seltener es auftritt, wobei allerdings $p(x_i)$ nicht Null werden darf.

Daraus folgt in anschaulicher Weise die Definition der Entropie:

$$H = \sum_{i=1}^{N} p(x_i) \cdot \mathrm{ld}\left(\frac{1}{p(x_i)}\right) \quad [\mathrm{bit}] \tag{3.1}$$

Aus (3.1) ergibt sich für den wichtigen Sonderfall einer binären Quelle unter der Voraussetzung, daß $p(x_{1,2})$=1/2 ist:

$$H_B = \sum_{i=1}^{2} p(x_i) \cdot \mathrm{ld}\left(\frac{1}{p(x_i)}\right) = \frac{1}{2}\mathrm{ld}(2) + \frac{1}{2}\mathrm{ld}(2) = 1\,\mathrm{bit} \tag{3.2}$$

(3.2) stellt die Definition der kleinsten Informationseinheit dar.

Allgemein gilt, daß die Entropie einer Quelle dann maximal wird, wenn die Auftrittswahrscheinlichkeit ihrer Zeichen gleich ist. Am Beispiel der binären Quelle läßt sich das einfach zeigen.

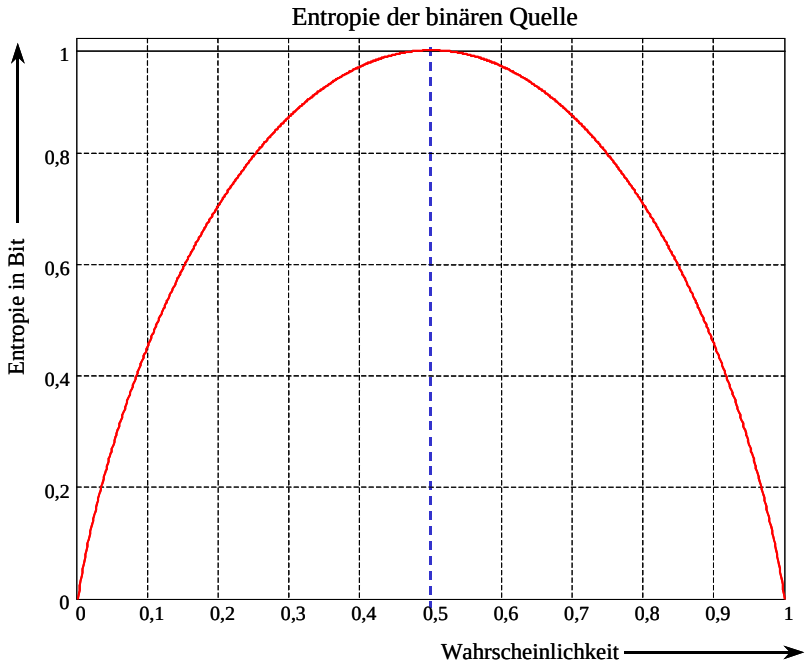


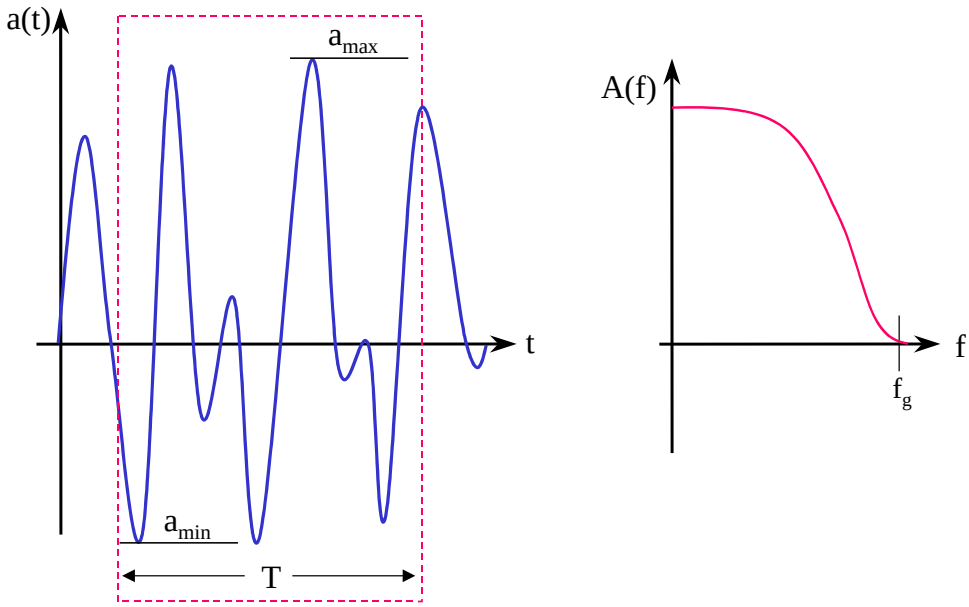
Bild 3.1: Entropie als Funktion der Wahrscheinlichkeit

Man sieht in Bild 3.1 sofort, daß das Maximum für $p(x_i)$=1/2 auftritt. Das läßt sich auch analytisch aus (3.2) ermitteln, wenn man dort z.B. $p(x_i)=\xi$ setzt, die Gleichung

$$\frac{-\xi}{\ln 2}\cdot\ln(\xi) + \frac{-(1-\xi)}{\ln 2}\cdot\ln(1-\xi)$$

nach ξ differenziert und das Ergebnis Null setzt. Man erhält ξ=1/2.

3.2 Verarbeitung zeit- und wertkontinuierlicher Signale



a_{max} und a_{min} haben physikalische Grenzen: a_{max} bei der Versorgungsspannung und a_{min} beim Hintergrundrauschen.

Das bedeutet, daß in einem technischen System nur eine endliche Anzahl Z von Amplitudenstufen in einem Bereich $|a_{max}-a_{min}|$ unterscheidbar ist.

Bild 3.2: Zeit- und Frequenzdarstellung eines Analogsignals

Unter der Annahme, daß jede der Amplitudenstufen mit gleicher Wahrscheinlichkeit auftritt, kann die Entropie der Signalquelle einfach berechnet werden:

$$H_{AS} = \mathrm{ld}(Z) \tag{3.3}$$

Um den Informationsgehalt eines Signalausschnitts der Dauer T zu bestimmen, wird die Zahl der Abtastwerte, die dem Signal innerhalb der Dauer T entnommen werden muß, benötigt. Sie wird durch das Shannon‘sche Abtasttheorem bestimmt, das besagt, daß die Abtastwerte mindestens mit dem Doppelten der oberen Grenzfrequenz f_g zu nehmen sind. Nur dann ist eine fehlerfreie Rekonstruktion des Analogsignals aus den Abtastwerten möglich. Innerhalb der Dauer T fallen somit $2f_gT$ Abtastwerte an, so daß man insgesamt

$$H_{AST} = 2 \cdot f_g \cdot T \cdot \mathrm{ld}(Z) \quad \text{bit} \tag{3.4}$$

für Informationsgehalt des Signalausschnitts erhält. Die erforderliche Datenrate zum Übertragen eines entsprechenden digitalen Signals wäre also H_{AST} bit/s.

3.2.1 Verfahren zur A/D- und D/A-Wandlung

Die Qualität digitaler Signalverarbeitungs- und Kommunikationssysteme wird wesentlich durch die Genauigkeit des eingangsseitigen Digitalsignals bestimmt. Die schwerwiegendsten Fehler passieren in der Regel bei der Analog/Digitalwandlung. Sie sind meist irreversibel, d.h. auch durch noch so aufwendige Signalverarbeitung nicht mehr korrigierbar.

Für die A/D-Wandlung sind zahlreiche Verfahren mit einer Reihe von Varianten bekannt. An dieser Stelle werden nur die wichtigsten grundlegend behandelt.

A/D-Wandler-Prinzipien:

- Integrierende Verfahren (Zählmethode über alle Amplitudenstufen)
- Sukzessive Approximation (schrittweise Annäherung in Zweierpotenzstufen)
- Flash-Wandlung (Direktwandlung in einem Schritt, mit sehr vielen Komparatoren)
- Pipelining-Prinzip (vereint Geschwindigkeit und hohe Auflösung)
- Sigma-Delta-Prinzip (sehr hohe Auflösung bei geringem Aufwand auf der Analogseite)

Die meisten A/D-Wandler beinhalten einen D/A-Wandler in einem Rückführungszweig, so daß die Genauigkeit wesentlich durch die Qualität dieses Bausteins mitbestimmt wird. Beim Flash- und Pipelining-Wandler ist die D/A-Komponente nicht auf den ersten Blick erkennbar, weil sie verteilt realisiert ist.

Einfache D/A-Wandler sind

- der R/2R-Wandler, dessen Kernstück ein einfaches Widerstandsnetzwerk ist
- der Pulsweitenmodulator, bei dem ein Analogwert durch das Tastverhältnis einer Rechteckfolge bestimmt wird

3.2.2 Abtastung

Die Aufgabe der A/D-Wandlung ist es, aus einem zeit- und wertekontinuierlichen Analogsignal $h_a(t)$ ein zeit- und wertdiskretes Abbild $h(n)$ zu gewinnen. Die Zeitdiskretisierung, d.h. der Abtastvorgang wird beschrieben durch

$$\hat{h}_a(t) = h_a(t) \cdot \sum_{n=-\infty}^{\infty} \delta(t - nT) = \sum_{n=-\infty}^{\infty} h(nT) \cdot \delta(t - nT) \qquad (3.5)$$

Die Funktion δ(t) in (3.5) repräsentiert den Diracimpuls, der das theoretische Modell des idealen Abtasters verkörpert. Seine Höhe geht gegen Unendlich, während die Breite gegen Null strebt und die gleich Eins ist. Obwohl die physikalische Vorstellung beim Diracimpuls versagt, ist er für analytische Berechnungen hervorragend geeignet. Die Summen in (3.5) beschreiben unendliche Dirac-Impulsfolgen, wobei die Impulse in äquidistanten Abständen T auftreten. Anschaulich hat man einen Impulskamm vor sich, der dem Signal $h_a(t)$ zu Zeitpunkten $t = nT$ Proben entnimmt. In Lehrbüchern zur Systemtheorie wird als Symbol für eine unendliche Dirac-Impulsfolge oft der russische Buchstabe III verwendet.

$\hat{h}_a(t)$ in (3.5) ist somit nur zu den Zeitpunkten $t = nT$ von Null verschieden. T wird als Abtastintervall oder $f_A=1/T$ als Abtastfrequenz bezeichnet. Durch die Multiplikation in (3.5) werden den Diracimpulsen Gewichte verliehen, die den Funktionswerten von $h_a(t)$ an den Stellen $t = nT$ entsprechen, so daß nun für $\hat{h}_a(t)$ nach Abtastung $\hat{h}_a(nT)$ geschrieben werden kann. In der Praxis läßt man – sofern Mißverständnisse ausgeschlossen werden können – T weg und schreibt vereinfacht $\hat{h}_a(n)$.

Mit $\hat{h}_a(n)$ hat man ein zeitdiskretes aber noch wertekontinuierliches Signal vor sich, das nachfolgend einer Quantisierung unterzogen werden muß, um schließlich das gewünschte zeit- und wertdiskrete Digitalsignal $h(n)$ zu erhalten – s. Bild 3.3.

Nach den Aussagen der Systemtheorie, ist es wichtig, das Abtastraster hinreichend eng zu wählen. In der westlichen Literatur wurde die erstaunliche Erkenntnis, daß nur zwei Proben pro Periode der höchsten vorkommenden Frequenz eines Signals benötigt werden, um es vollständig zu erfassen, erstmalig 1948 von C.E. Shannon als „Abtasttheorem“ veröffentlicht. Spätere Recherchen förder-

ten jedoch zutage, daß bereits 1933 der russische Wissenschaftler Vladimir Kotelnikov eine exakte mathematische Darstellung des Abtasttheorems in einem sowjetischen Konferenzbericht publiziert hatte, der allerdings der westlichen Welt bis Mitte der neunziger Jahre des letzten Jahrhunderts unbekannt blieb. Im Jahre 1999 erhielt Prof. Kotelnikov, Mitglied der russischen Akademie der Wissenschaften, im Alter von 91 Jahren als späte Anerkennung seiner Leistung in München den Eduard-Rhein-Grundlagenpreis.

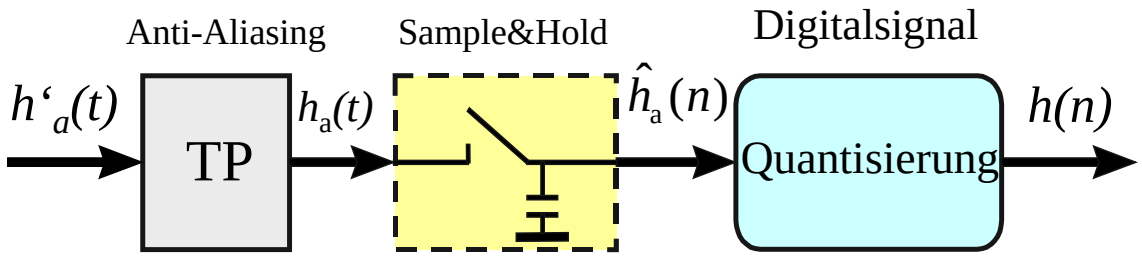


Bild 3.3: Allgemeine Übersicht der A/D-Wandlungsschritte

Die Herleitung des Abtasttheorems erfolgt in übersichtlicher Weise mittels der Fouriertransformation der unendlichen Dirac-Impulsfolge. Mit der erwähnten Schreibweise unter Verwendung des russischen Buchstabens Ш (sch) erhält man zunächst

$$Ш(t) = \sum_{n=-\infty}^{\infty} \delta(t-n). \tag{3.6}$$

Für die Fouriertransformation folgt dann

$$Ш(t) \quad \circ\!\!-\!\!\bullet \quad Ш(f) \tag{3.7}$$

Offenbar hat man es bei diesem Transformationspaar mit einer ’selbstabbildenden’ Funktion zu tun. Ein anderes Beispiel einer solchen Funktion ist die Gaußverteilung ($f(x) = e^{-x^2}$).

Der Beweis für (3.7) ist in den meisten Lehrbüchern der Nachrichten- oder Systemtheorie zu finden. Hier wird eine kurze anschauliche Darstellung gegeben. Die Gleichung (3.7) kann etwas erweitert in der Form

$$Ш(t) = \sum_{n=-\infty}^{\infty} \delta(t-n) \quad \circ\!\!-\!\!\bullet \quad Ш(f) = \sum_{n=-\infty}^{\infty} \delta(f-n) = \sum_{n=-\infty}^{\infty} \mathrm{e}^{-\mathrm{j}2\pi f n} \tag{3.8}$$

geschrieben werden. Wie man sieht, läßt sich die unendliche Dirac-Impulsfolge als eine unendliche Summe komplexer sinusförmiger (harmonischer) Schwingungen darstellen. Da die Summation über ein symmetrisches Intervall (–∞...+∞) erfolgt, verschwinden die ungeraden, d.h. die Sinusterme und für die verbleibenden Cosinusterme gilt

$$Ш(f) = \sum_{n=-\infty}^{\infty} \mathrm{e}^{-\mathrm{j}2\pi n f} = 1 + 2 \cdot \sum_{n=1}^{\infty} \cos(2\pi n f), \tag{3.9}$$

wobei für n=0 noch die Eins berücksichtigt werden muß. Mit dieser Vereinfachung werden nun schrittweise Simulationen durchgeführt, wobei die Zahl n der berücksichtigten Schwingungen von vier auf tausend erhöht wird.

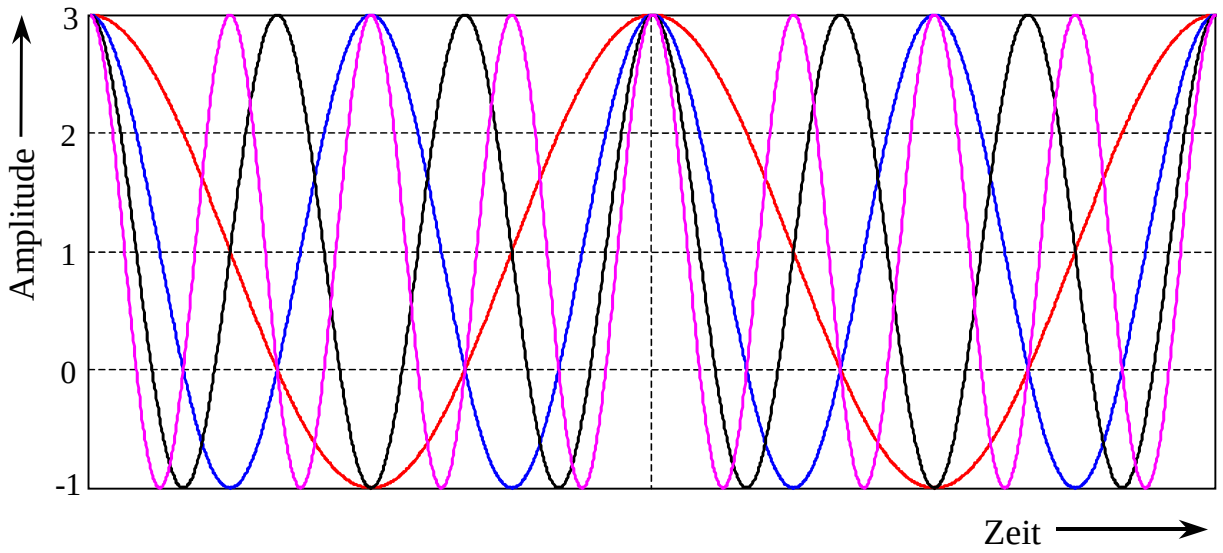


Bild 3.4: Zeitverläufe der einzelnen Schwingungen für n=4

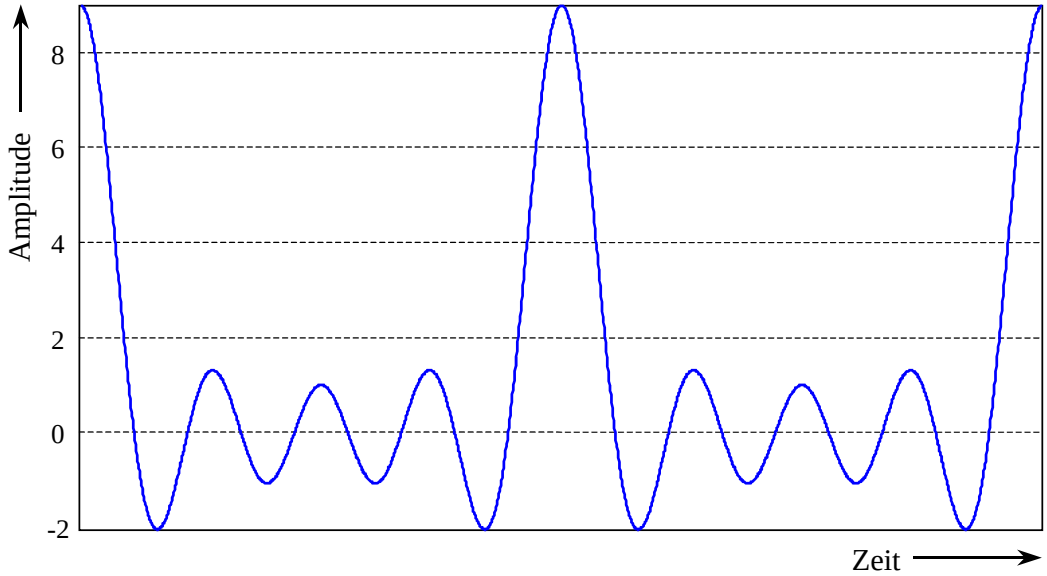


Bild 3.5: Summation der vier Schwingungen

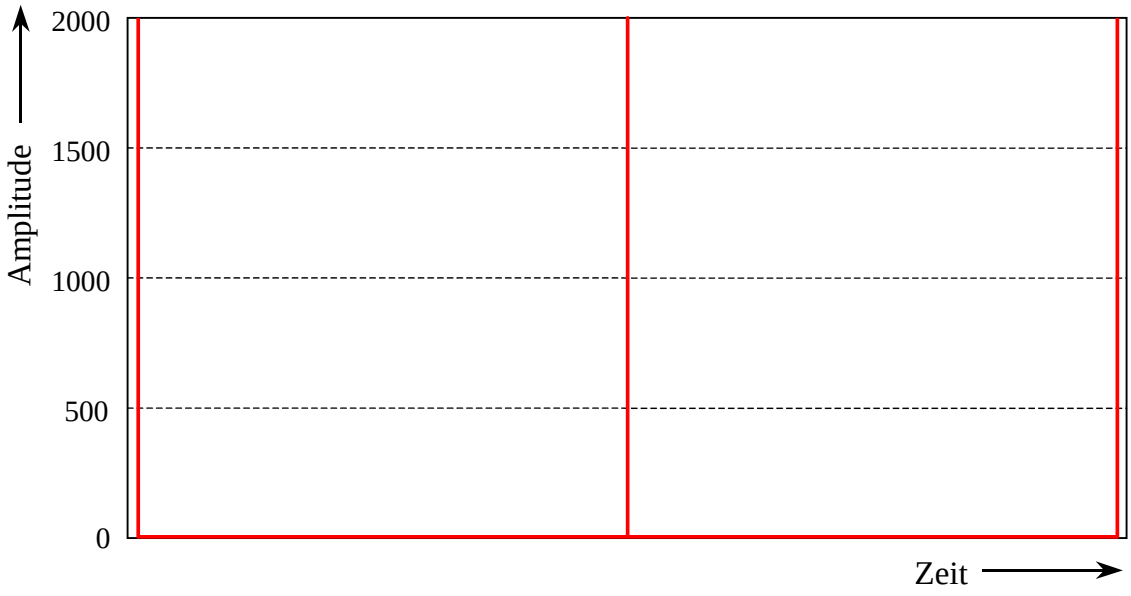


Bild 3.6: Summation von 1000 Schwingungen

Wie Bild 3.5 zeigt, bilden sich bereits für n=4 deutliche 'Nadelimpulse' heraus, die bei weiterer Vergrößerung von n mehr und mehr die Form eines Diracimpulses annehmen. Das demonstriert Bild 3.6 schon sehr eindrucksvoll für n=1000. Die Höhe ist allerdings noch nicht Unendlich, sondern beträgt $2n$+1=2001 in diesem Beispiel, während die Breite aufgrund der grafischen Auflösungen bereits gegen Null zu gehen scheint.

In der Praxis genügt die „unskalierte" Darstellung von Dirac-Impulsfolgen gemäß (3.6) ... (3.8) nicht, sondern es muß möglich sein, ein beliebiges Abtastraster T zu einzuführen. Das gelingt in der folgenden Form:

$$\text{Ш}\left(\frac{t}{T}\right) = \sum_{n=-\infty}^{\infty} \delta\left(\frac{t}{T} - n\right) \tag{3.10}$$

Mit der Dehnungseigenschaft des Diracimpulses gilt

$$\delta(\mathrm{b}t) = \frac{1}{|\mathrm{b}|} \cdot \delta(t) \quad \text{für eine beliebige Konstante b} \tag{3.11}$$

Angewandt auf (3.10) bedeutet das

$$\text{Ш}\left(\frac{t}{T}\right) = \sum_{n=-\infty}^{\infty} \delta\left(\frac{1}{T} \cdot (t - nT)\right) = |T| \cdot \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{3.12}$$

Im weiteren interessiert jetzt die Fouriertransformation von (3.12), wobei das Ähnlichkeitstheorem der FT (Zeitskalierung)

$$h(\mathrm{b}t) \circ\!\!-\!\!\bullet \frac{1}{|\mathrm{b}|} \cdot H\left(\frac{f}{\mathrm{b}}\right), \text{ mit einer beliebigen Konstante b} \tag{3.13}$$

zu beachten ist. Man erhält

$$\begin{aligned} \text{Ш}\left(\frac{t}{T}\right) \quad \circ\!\!-\!\!\bullet \quad & |T| \cdot \text{Ш}(fT) = |T| \cdot \sum_{n=-\infty}^{\infty} \delta(fT - n) = |T| \cdot \sum_{n=-\infty}^{\infty} \delta\left(T\left(f - \frac{n}{T}\right)\right) = \\ & |T| \cdot \frac{1}{|T|} \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{T}\right) = \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{T}\right) \end{aligned} \tag{3.14}$$

Um bei realen Signalen das Abtasttheorem erfüllen zu können, ist in der Regel ein Tiefpaßfilter -(TP) in Bild 3.3 – erforderlich, das dafür sorgt, daß sich oberhalb einer Grenzfrequenz f_g keine Spektralanteile mehr befinden. Der Tiefpaß wird auch Anti-Aliasingfilter genannt, ein Begriff, der sich bei der Betrachtung des Spektrums des abgetasteten Signals selbst erklärt. In der Praxis erfordert die Realisierung des Tiefpasses gelegentlich einigen Aufwand.

Mit den obigen Vorbereitungen kann jetzt das Spektrum $\hat{H}_\mathrm{a}(f)$ des Abtastsignals $\hat{h}_\mathrm{a}(t)$ durch Anwendung der Fouriertransformation auf (3.5) bestimmt werden. Nach (3.12) und (3.14) gilt $\sum_{n=-\infty}^{\infty} \delta(t - nT) = \frac{1}{|T|} \text{Ш}\left(\frac{t}{T}\right) \circ\!\!-\!\!\bullet \frac{1}{|T|} \cdot \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{T}\right)$, so daß sich

$$\hat{H}_\mathrm{a}(f) = H_\mathrm{a}(f) * \frac{1}{|T|} \cdot \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{T}\right) = \frac{1}{|T|} \sum_{n=-\infty}^{\infty} H_\mathrm{a}\left(f - \frac{n}{T}\right) \tag{3.15}$$

ergibt. Wie man aus (3.15) ersieht, bewirkt die Abtastung mit einer unendlichen Dirac-Impulsfolge im Zeitbereich eine Faltung des Spektrums $H_a(f)$ im Frequenzbereich mit einer ebenfalls unendlichen Dirac-Impulsfolge. Das Ergebnis ist ein unbegrenztes Spektrum $\hat{H}_a(f)$, in dem das Grundspektrum $H_a(f)$ unendlich oft wiederholt vorkommt und jeweils symmetrisch bei ganzzahligen Vielfachen der Abtastfrequenz f_A positioniert ist, also bei 0, $\pm f_A$, $\pm 2f_A$, ...

Wäre $H_a(f)$ nicht exakt bandbegrenzt, würden in Bild 3.7 Überlappungen (Aliasing) auftreten, mit der Folge, daß das Originalsignal aus dem Abtastsignal nicht mehr rekonstruierbar wäre. Wie Bild 3.7 zeigt, gelingt dies mit dem eingezeichneten idealen Rekonstruktionstiefpaß.

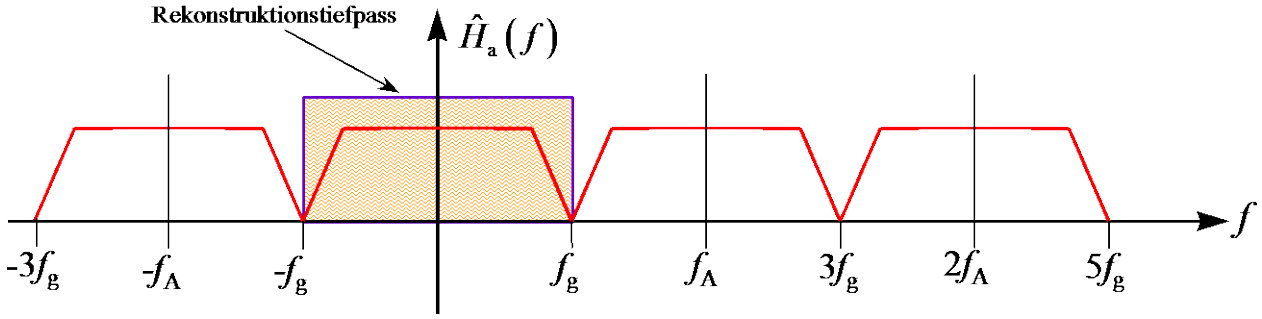
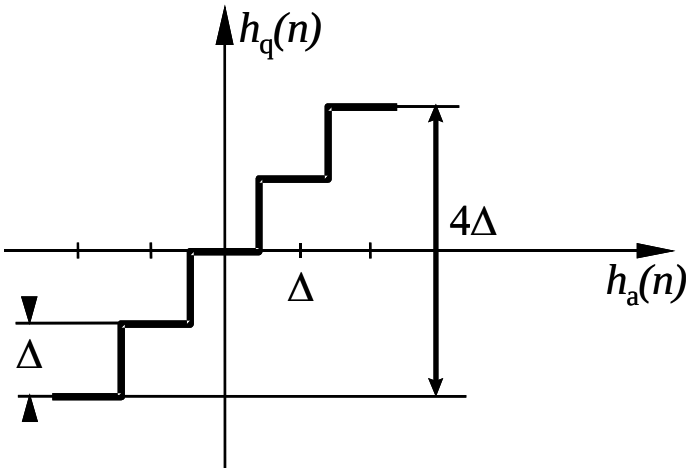


Bild 3.7: Ausschnitt aus dem Spektrum eines bandbegrenzten Signals nach der Abtastung

Des weiteren ist in Bild 3.7 leicht erkennbar, daß mit schnellerer Abtastung, d.h. $f_A > 2f_g$, ein Auseinanderrücken der Spektren erfolgt, so daß Lücken entstehen. Man spricht dann von Überabtastung oder Oversampling. Obwohl Überabtastung auf den ersten Blick nach Verschwendung aussieht lassen sich damit vielfältige Vorteile gewinnen. Der Sigma-Delta-Wandler, der weiter unten ausführlich behandelt wird, ist ein hervorragendes Beispiel dafür.

3.2.3 Quantisierung

Während die Einhaltung des Abtasttheorems die Möglichkeit zur vollständigen, fehlerfreien Rekonstruktion eines Signals aus den Abtastwerten garantiert, bedeutet der nun folgende Schritt der Quantisierung eine irreversible Verschlechterung der Signalqualität. Beim Quantisieren muß man ein Wertekontinuum auf diskrete Stufen abbilden, wobei alle Information über Zwischenwerte verloren geht.



Es ist offensichtlich, daß eine Grobstufung zu schlechteren Ergebnissen führt als eine Feinstufung. In der Praxis wird ein Maß benötigt, mit dem man die Qualität eines Wandlers beurteilen kann. Dabei hat sich die Angabe eines Signal-Störverhältnisses bewährt, das im folgenden hergeleitet wird. Bild 3.8 beschreibt einen symmetrischen Quantisierer, der das zeitdiskrete und wertkontinuierliche Eingangssignal $h_a(n)$ in ein gestuftes, wertdiskretes Ausgangssignal $h_q(n)$ umsetzt.

Bild 3.8: Symmetrische Quantisierungskennlinie

Im dargestellten Beispiel mit vier Amplitudenstufen der Höhe Δ erkennt man, daß sich z.B. ein Sinussignal mit der maximalen Amplitude $A_{max}=2\cdot\Delta$ quantisieren ließe, was allerdings eine sehr grobe Stufung bedeuten würde.

Für praktisch sinnvolle Anwendungen muß die Kennlinie nach Bild 3.8 auf eine um vieles größere Zahl k von Amplitudenstufen erweitert werden, wobei dann gilt

$$k = \frac{2 \cdot A_{\max}}{\Delta} + 1, \tag{3.16}$$

$$\text{bzw. } A_{\max} = \frac{k-1}{2} \cdot \Delta \tag{3.17}$$

Aus der Zahl der Amplitudenstufen läßt sich nun die Auflösung in Bit angeben:

$$\frac{Z}{\text{bit}} = \text{ld}(k) = \text{ld}\left\{\frac{2A_{\max}}{\Delta} + 1\right\} \approx \text{ld}\left\{\frac{2A_{\max}}{\Delta}\right\} \tag{3.18}$$

Quantisierungsfehler:

Bei Betrachtung der Kennlinie in Bild 3.8 wird klar, daß in jeder Stufe ein Fehler beim Übergang vom Kontinuierlichen zum Diskreten vorkommen kann, der jedoch auf eine Stufenhöhe Δ begrenzt ist. D.h. es befinden sich alle Fehler innerhalb eines Intervalls der Breite Δ. Für die folgenden Untersuchungen ist eine Aussage über die Verteilung der Fehler über diesem Intervall von Bedeutung. Man benötigt die Verteilungsdichtefunktion, die angibt, mit welcher Wahrscheinlichkeit $p_q(\varepsilon)$ der Quantisierungsfehler einen Wert ε innerhalb des Intervalls $-\Delta/2...+\Delta/2$ annimmt. Diese Wahrscheinlichkeit ist sicher abhängig vom Signalverlauf, der aber in das gesuchte Maß zur Qualitätsbewertung so wenig wie möglich eingehen sollte. Um den Ansatz weitgehend allgemein zu halten – d.h. keine Signalspezifikation vorzugeben – nimmt man eine Gleichverteilung gemäß

$$p_q(\varepsilon) = \frac{1}{\Delta}\text{rect}\left(\frac{\varepsilon}{\Delta}\right) \tag{3.19}$$

an – s. Bild 3.9. Die Funktion rect(·) beschreibt ein Rechteck mit der Höhe und der Fläche Eins, das sich im Koordinatensystem von –1/2 bis +1/2 erstreckt.

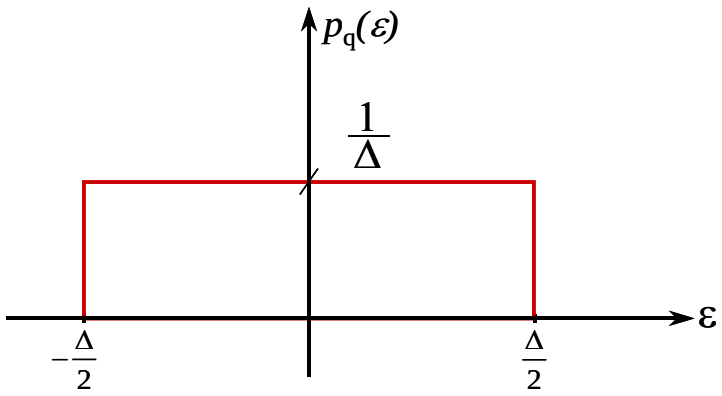


Bild 3.9:Verteilungsdichtefunktion des Quantisierungsfehlers

Wenn die Verteilungsdichtefunktion des Quantisierungsfehlers bekannt ist, kann das 2. Moment der Funktion berechnet werden, das der Leistung N_q des Quantisierungsrauschens entspricht.

Die Berechnung ist bei der vorliegenden Form der Verteilungsdichtefunktion einfach und ergibt

$$N_q = \int_{-\infty}^{\infty} p_q(\varepsilon) \cdot \varepsilon^2 d\varepsilon = \frac{1}{\Delta}\int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} \varepsilon^2 d\varepsilon = \frac{\Delta^2}{12}. \tag{3.20}$$

In der Praxis kann man mit dem Absolutwert des Quantisierungsrauschens wenig anfangen. Erst wenn man ihn ins Verhältnis zu einer Nutzsignalleistung setzt, erhält man das aussagekräftige

Signal-Störverhältnis. Zur Bestimmung einer Nutzsignalleistung nehmen wir an, daß ein sinusförmiges Signal mit der Amplitude A_{max} vorliegt, das die Quantisierungskennlinie voll ausfährt. Ein solches Signal hat die normierte – d.h. an einem Widerstand von 1Ω auftretende – Leistung

$$S_s = \frac{A_{max}^2}{2} \tag{3.21}$$

Mit (3.18) erhalten wir

$$A_{max} = \frac{1}{2} \cdot 2^Z \cdot \Delta \tag{3.22}$$

und damit

$$S_s = \frac{A_{max}^2}{2} = \frac{1}{8} \cdot 2^{2Z} \cdot \Delta^2 . \tag{3.23}$$

Da auch bei der Berechnung der Leistung des Quantisierungsrauschens in (3.20) implizit ein Bezugswiderstand von 1 Ω vorausgesetzt wurde, läßt sich jetzt das gesuchte Signal-Störverhältnis

$$\frac{S_s}{N_q} = \frac{1}{8} \cdot 2^{2Z} \cdot \frac{\Delta^2}{\Delta^2/12} = \frac{3}{2} \cdot 2^{2Z} \tag{3.24}$$

angeben. Um die Schreibweise zu vereinfachen, wird anstelle des linearen Signal-Störverhältnisses in der Praxis meist ein logarithmierter Wert in dB angegeben. Zur Unterscheidung wird dafür im weiteren der Begriff Störabstand *a* verwendet. Es gilt der folgende Zusammenhang:

$$\frac{a}{\text{dB}} = 10\lg\left(\frac{S_s}{N_q}\right) = 10\lg(1{,}5) + 20 \cdot Z \cdot \lg(2) = 1{,}76 + 6{,}02 \cdot Z \tag{3.25}$$

Mit diesem Ergebnis lassen sich in sehr einfacher Weise A/D-Wandler verschiedener Auflösung (d.h. Bitzahl *Z*) im Hinblick auf ihr Quantisierungsrauschen miteinander vergleichen. So beträgt der Störabstand für einen 8bit-Wandler z.B. rund 50dB, während man mit 16 bit bereits 98 dB erreicht. Mit diesen Zahlen kann der Ingenieur in der Praxis etwas anfangen, weil er z.B. weiß, daß das menschliche Gehör einen Dynamikumfang in der Größenordnung von 95dB hat. Für eine qualitativ hochwertige Audiosignalverarbeitung müssen daher mindestens 16 bit zur Verfügung stehen. Werden dagegen nur 8 bit verwendet, nimmt man ein kräftiges, störendes Hintergrundrauschen wahr, das den Genuß klassischer Musik auf jeden Fall verdirbt.

3.2.4 A/D-Wandelverfahren

3.2.4.1 Integrierende Verfahren (Zählmethode über alle Amplitudenstufen)

Bild 3.10 zeigt den prinzipiellen Aufbau eines Wandlers nach dem Zählverfahren. Der Name kommt von dem digitalen Zähler, der ein zentrales Element des Wandlers ist. Zu Beginn der Wandlung steht der Zähler auf Null und alle *N* Bits des digitalen Ausgangssignals sind ebenfalls Null.

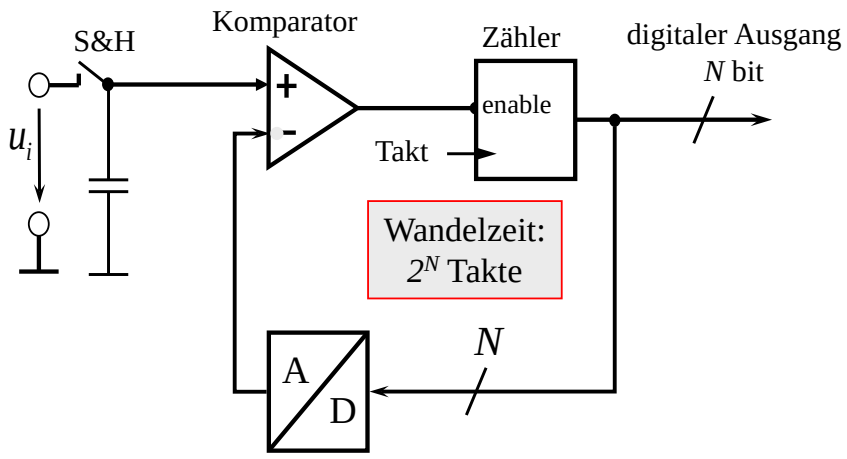


Bild 3.10: Integrierender A/D-Wandler nach dem Zählverfahren

Der D/A-Wandler im Rückführungszweig liefert deshalb den Analogwert Null and den –Eingang des Komparators, während der +Eingang die zu wandelnde Spannung u_i erhält, die hier aus Übersichtsgründen als positiv angenommen wird. Solange u_i größer ist als der über den D/A-Wandler rückgeführte Wert, läuft der Zählvorgang weiter, d.h. mit jedem Taktimpuls erhöht sich der Zählerinhalt um Eins. Ebenso wächst der Analogwert am –Eingang des Komparators, dessen Ausgang das Freigabesignal (Enable) für den Zähler liefert (hier ein logischer „1“-Pegel). Sobald der rückgeführte Analogwert die Größe der Eingangsspannung u_i am Komparator erreicht, kippt dessen Ausgang auf logisch „0“ und stoppt den Zähler. Der Wandelvorgang ist beendet und der Digitalwert steht am Zählerausgang bereit. A/D-Wandler nach dem Zählverfahren ermöglichen heute Wandelraten bis zu einigen hundert kHz, bei Auflösungen bis ca. 12 bit. Sie sind nicht schnell, weil für einen Wandelvorgang 2^N Takte für N bit Auflösung erforderlich sind (bei 12 bit z.B. 2048). Wegen des einfachen Aufbaus sind sie sehr preisgünstig. Man findet sie deshalb in Anwendungen, wo es um die Digitalisierung von langsamen Signalen einfacher Sensoren (Temperatur, Position, Feuchte, Druck etc.) geht. Es gibt vielfältige Varianten des Prinzips nach Bild 3.10. Am bekanntesten ist das „Zweirampenverfahren“ (engl.: Dual Slope), mit dem es gelingt, das Wandelergebnis von Bauteiltoleranzen, Offsetspannungen und Streuungen der Taktfrequenz weitgehend unabhängig zu machen.

3.2.4.2 Sukzessive Approximation (schrittweise Annäherung in Zweierpotenzstufen)

Bei diesem Verfahren wird mit verhältnismäßig geringem Zusatzaufwand eine erhebliche Beschleunigung des Wandelvorganges erzielt, ohne dabei Genauigkeit einzubüßen. Anstelle des Zählers tritt jetzt ein „sukzessives Approximationsregister“, in dem die einzelnen Bits des digitalen Ausgangsignals gesetzt oder rückgesetzt werden. Der Ablauf dieser Vorgänge wird über einen endlichen Automaten (FSM ≡ Finite State Machine) gesteuert.

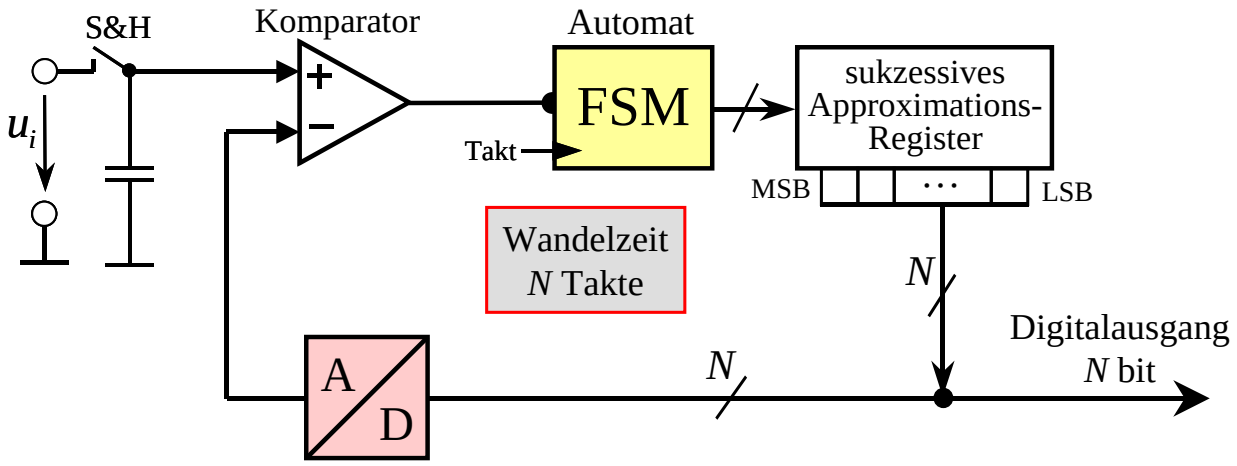


Bild 3.11: Prinzip des SAR-Wandelverfahrens

Eingangsseitig hat man in Bild 3.11 den gleichen Aufbau wie inBild 3.10. Das Ausgangssignal des Komparators bewirkt aber jetzt nicht die Freigabe eines Zählers, sondern signalisiert der FSM, ob der rückgeführte D/A-gewandelte Wert des Digitalsignals das Eingangssignal u_i übersteigt oder nicht.

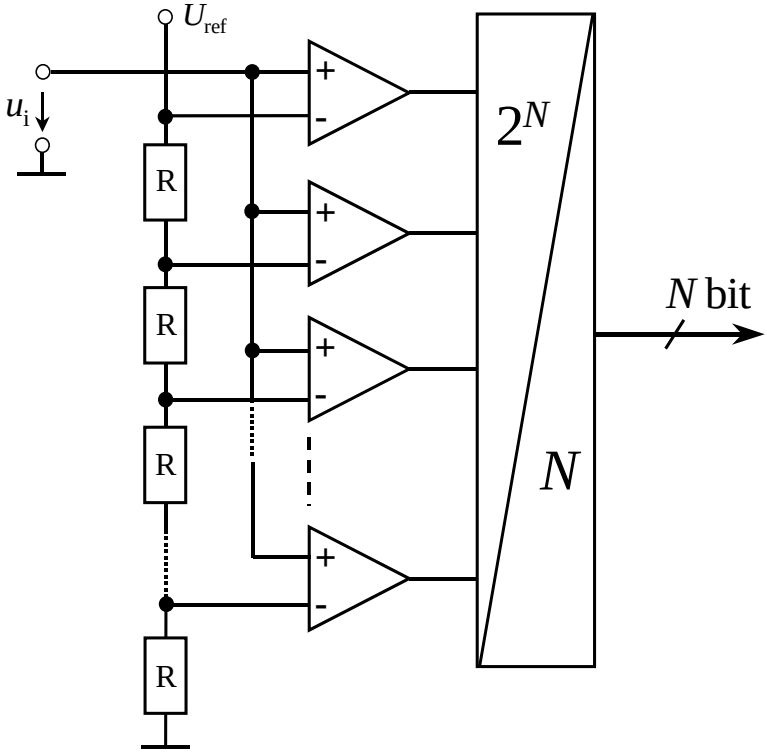
Zu Beginn einer Wandlung sind alle Bits des SAR auf Null, so daß der Komparator der FSM mit einem „1“-Pegel signalisiert, daß u_i größer als der aktuelle Digitalwert ist. Die FSM setzt daraufhin das MSB auf „1“. Bleibt der Komparatorausgang danach auf „1“, wird das nächste, niederwertigere Bit des SAR von der FSM gesetzt. Wenn u_i den vollen Wertebereich des Wandlers ausschöpft, ist der weitere Ablauf sehr einfach: Die FSM setzt Bit für Bit im SAR bis zum LSB und die Wandlung ist beendet. Kippt jedoch zwischendurch der Komparator, bedeutet dies, daß der

rückgeführte Wert zu groß ist. Die FSM nimmt dann das gerade gesetzte Bit wieder zurück und setzt das nächste niederwertigere. Auch hierbei ist die Wandlung mit Erreichen des LSB beendet. Der beschriebene Ablauf kann auch leicht für den Extremfall, wenn u_i gerade nur so groß wie das LSB ist, nachvollzogen werden.

Da in allen Fällen in jedem Taktschritt ein Bit des Ausgangssignals festgelegt wird, ist stets nach N Takten ein Wandelvorgang abgeschlossen. Für eine Auflösung von 12 bit werden demnach exakt 12 Taktzyklen benötigt, im Gegensatz zu 2048 beim Zählverfahren. Trotz dieses enormen Zeitvorteils hat die Wandelrate auch beim SAR-Verfahren Grenzen, denn die maximale Taktfrequenz für die FSM wird ungefähr durch den Kehrwert der Summe der Verzögerungszeiten von FSM, SA-Register, D/A-Wandler und Komparator vorgegeben. SAR-Wandler werden für Wandelraten von etwa 500 kHz bis über 20 MHz bei Auflösungen von 12...16 bit eingesetzt. Sie decken das größte Anwendungsspektrum ab.

3.2.4.3 Flash-Wandlung (Direktwandlung in einem Schritt mit vielen Komparatoren)

Der Flash-Wandler hat seinen Namen wegen der hohen Geschwindigkeit, die im wesentlichen von der Reaktionszeit *eines einzigen* Komparators und der Codierlogik am Ausgang bestimmt wird. Seine Funktionsweise ist einfach zu verstehen, die Herstellung jedoch extrem aufwendig. Wie Bild 3.12 zeigt, wird für jede Amplitudenstufe ein eigener Komparator benötigt, der an seinem +Eingang die zu wandelnde Eingangsspannung u_i und am –Eingang in feiner Abstufung die Referenz erhält. Für eine Auflösung von 8 bit sind somit 256 Komparatoren und ebenso viele Vergleichsspannungen nötig. Nach dem Anlegen von u_i ist der Wandelvorgang nach einer Komparator-Durchlaufzeit beendet, da alle Komparatoren parallel arbeiten. Zur Ausgabe gelangt aber nur das Signal (logischer „1“-Pegel) desjenigen Komparators mit der höchsten Vergleichsspannung. Alle darunter liegenden liefern selbstverständlich auch „1“-Pegel, die aber ignoriert werden müssen. Die darüber liegenden liefern „0“. Die Codierlogik hat somit die Aufgabe, die Zuordnung eines aus 2^N Signalen der Breite 1-bit auf ein N bit breites Datenwort vorzunehmen und kann recht umfangreich werden. Auch hier kann durch Parallelität die Durchlaufzeit minimiert werden.



Flash-Wandler erlauben bei einer Auflösung von 8 bit Wandelraten von mehreren hundert MHz. Für die Kommunikationstechnik gibt es Ausführungen mit 6 bit, die Signale bis über 1 GHz verarbeiten können, d.h. ihre Wandelrate muß größer als 2 GHz sein.

Bild 3.12: Das Flash-Wandler-Prinzip

3.2.4.4 Das Pipelining-Prinzip

Das in Bild 3.13 dargestellte Pipelining-Prinzip vereinigt die Geschwindigkeit einer Direktwandlung (Flash) mit der schrittweisen Erzeugung des digitalen Ausgangssignals. A/D-Wandler nach diesem Prinzip ermöglichen Wandelraten im Bereich 50...150 MHz, bei Auflösungen von 10...14 bit.

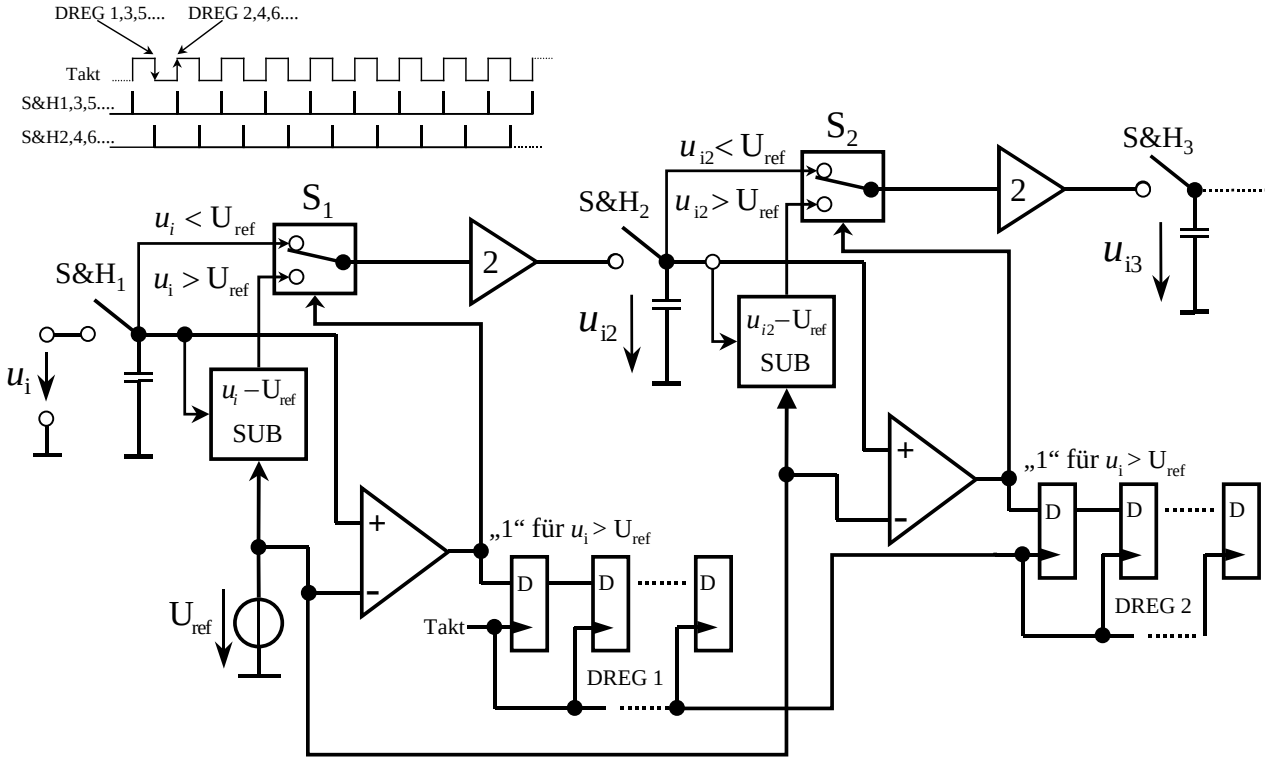


Bild 3.13: Das Pipelining-Prinzip zur schnellen und hochauflösenden A/D-Wandlung

Pipelining ist in der gesamten Digitaltechnik von großer Bedeutung und wurde schon in mehreren Anwendungen wie ***Instruktions-Pipelining*** oder ***Pipeline im Parallelmultiplizierer*** betrachtet.

- Pipelining läuft stets nach Maßgabe eines Taktes in mehreren Schritten ab
- In jedem dieser Schritte wird eine bestimmte Funktion ***autonom*** durchgeführt
- Für die autonomen Funktionen sind voneinander ***unabhängige*** Komponenten nötig
- Die unabhängigen Komponenten verarbeiten ***parallel verschiedene*** Daten
- Mit ***jedem Taktschritt*** wird ***nach Füllung*** der Pipeline ein Ergebnis geliefert

Eine *n*-stufige Pipeline benötigt somit *n* autonome Funktionseinheiten, die möglichst gleich lange Durchlaufzeiten für Signale aufweisen sollten. Nach *n* Taktschritten ist die Pipeline gefüllt und liefert mit jedem Takt ein komplettes Ergebnis, obwohl zu dessen Erzeugung immer *n* Takte benötigt werden. Das ist möglich, da aufgrund der parallel vorhandenen Funktionseinheiten stets *n* verschiedene aufeinander folgende Operanden gleichzeitig in der Bearbeitung sind. Der Aufwand für Pipelining steckt also in der Parallelisierung, was aus Bild 3.13 sofort ersichtlich ist. Das gezeigte Beispiel beschreibt einen A/D-Wandler, der in *N* Stufen von je 1 bit Auflösung Ergebnisse mit *N* bit erzeugt. Es gibt zahlreiche Varianten, die sich z.B. durch höhere Auflösung (2 bit, 4 bit) in jeder Stufe und entsprechend verringerter Stufenzahl unterscheiden. Aus Übersichtsgründen behandeln wir hier nur 1 bit-Stufen.

In Bild 3.13 erkennt man am Eingang jeder Stufe ein Sample-and-Hold-Glied (S&H). Nach Maßgabe des Taktes nehmen diese S&Hs zeitversetzt Abtastwerte der Eingangsspannung u_i bzw. der Ausgangsspannungen u_{in} der Vorstufen. Wie das Impulsdiagramm links oben in Bild 3.13 andeu-

tet, werden die S&Hs der ungeraden Stufen jeweils mit einer Taktvorderflanke betätigt, während die geraden mit Rückflanken arbeiten. Auf diese Weise steht jeder Stufe rund eine halbe Taktperiode zur Verarbeitung eines Abtastwertes zur Verfügung, wenn ein Takt mit Tastverhältnis 1:1 verwendet wird. Man erkennt leicht, daß das nicht optimal ist. Ein Takt mit möglichst langer „1"-Phase und sehr kurzer „0"-Phase stellt mehr Verarbeitungszeit zur Verfügung.

Zur Funktionsbeschreibung wird die erste Stufe betrachtet und angenommen, daß $S\&H_1$ gerade mit einer Taktvorderflanke betätigt wurde. Der Abtastwert von u_i gelangt an den Subtrahierer (SUB), der die Referenzspannung $U_{ref}=1/2\cdot u_{imax}$ abzieht und das Ergebnis an den Schalter S_1 liefert. Gleichzeitig vergleicht der Komparator K_1 den Abtastwert von u_i mit der Referenz und liefert an seinem Ausgang einen logischen „1"-Wert, falls $u_i > U_{ref}$ ist, ansonsten liefert er „0". Dieser Digitalwert wird zum einen mit einer Taktrückflanke in das Register $DREG_1$ übernommen und betätigt zum anderen den Schalter S_1. Mit einem „1"-Pegel wird der Schalter in die untere Stellung gebracht, so daß der Wert u_i-U_{ref} auf den Eingang eines Verstärkers gelangt, der ihn exakt um den Faktor 2 verstärkt. $S\&H_2$ übernimmt nun mit der Taktrückflanke $2(u_i - U_{ref})$. Bei einem „0"-Pegel an K_1 wird der Schalter S_1 in die obere Stellung gebracht, so daß $2u_i$ in die nachfolgende Stufe übernommen wird. In das erste Flip-Flop von $DREG_1$ wird dann „0" eingeschrieben. Der beschriebene Ablauf findet in jeder Stufe gleichzeitig statt, wobei die Ausgangssignale von Stufe zu Stufe mit Zwei verstärkt weitergereicht werden, so daß z.B. die letzte Stufe $2^{N-1}(u_{i(N-1)}-U_{ref})$ erhält. In den Registern $DREG_i$, deren Länge von Stufe zu Stufe um Eins abnimmt, werden die Vergleichsergebnisse der Komparatoren K_i abgelegt und wandern mit jedem Takt nach rechts. Die letzte Stufe benötigt somit nur noch ein Flip-Flop. Im Beispiel nach Bild 3.13 ist zu beachten, daß in den ungeraden Stufen die Datenübernahme jeweils mit einer Taktrückflanke und in den geraden mit einer Vorderflanke erfolgen muß. Aus der jeweils letzten Stufe der Register $DREG_i$ kann nach kompletter Füllung der Pipeline mit jedem Takt ein fertig gewandelter Digitalwert ausgelesen werden.

Ein einfaches Zahlenbeispiel soll die einzelnen Schritte noch mal verdeutlichen:

$u_{imax}=10\text{ V} \Rightarrow U_{ref}=5\text{ V}$

$N=8$ (8bit Auflösung)

$u_i=6\text{ V}$

Um Schreibarbeit zu sparen, wird u_i als konstant angenommen, so daß jeder Wandelvorgang den gleichen Ausgangswert liefert; sonst müßte die folgende Tabelle 3.1 achtmal dargestellt werden.

Tabelle 3.1: Durchlauf der Ergebnisse durch die einzelnen Wandlerstufen

Stufe 1	Stufe 2	Stufe 3	Stufe 4	Stufe 5	Stufe 6	Stufe 7	Stufe 8
$u_i=6V$	$u_{i2}=2V$	$u_{i3}=4V$	$u_{i4}=8V$	$u_{i5}=6V$	$u_{i6}=2V$	$u_{i7}=4V$	$u_{i8}=8V$
$u_i>U_{ref}$	$u_{i2}<U_{ref}$	$u_{i3}<U_{ref}$	$u_{i4}>U_{ref}$	$u_{i5}>U_{ref}$	$u_{i6}<U_{ref}$	$u_{i7}<U_{ref}$	$u_{i8}>U_{ref}$
$u_{i2}=2\cdot 1V$	$u_{i3}=2\cdot 2V$	$u_{i4}=2\cdot 4V$	$u_{i5}=2\cdot 3V$	$u_{i6}=2\cdot 1V$	$u_{i7}=2\cdot 2V$	$u_{i8}=2\cdot 4V$	---
K_1: „1"	K_2: „0"	K_3: „0"	K_4: „1"	K_5: „1"	K_6: „0"	K_7: „0"	K_8: „1"

Tabelle 3.2: Schrittweiser Aufbau der Ergebnisse in den Registern der einzelnen Stufen

$DREG_1$	$DREG_2$	$DREG_3$	$DREG_4$	$DREG_5$	$DREG_6$	$DREG_7$	$DREG_8$
1000 0000	000 0000	00 0000	0 0000	0000	000	00	0
1100 0000	000 0000	00 0000	0 0000	0000	000	00	0
1110 0000	000 0000	00 0000	0 0000	0000	000	00	0
1111 0000	000 0000	00 0000	1 0000	0000	000	00	0
1111 1000	000 0000	00 0000	1 1000	1000	000	00	0
1111 1100	000 0000	00 0000	1 1100	1100	000	00	0
1111 1110	000 0000	00 0000	1 1110	1110	000	00	0
1111 1111	000 0000	00 0000	1 1111	1111	000	00	1

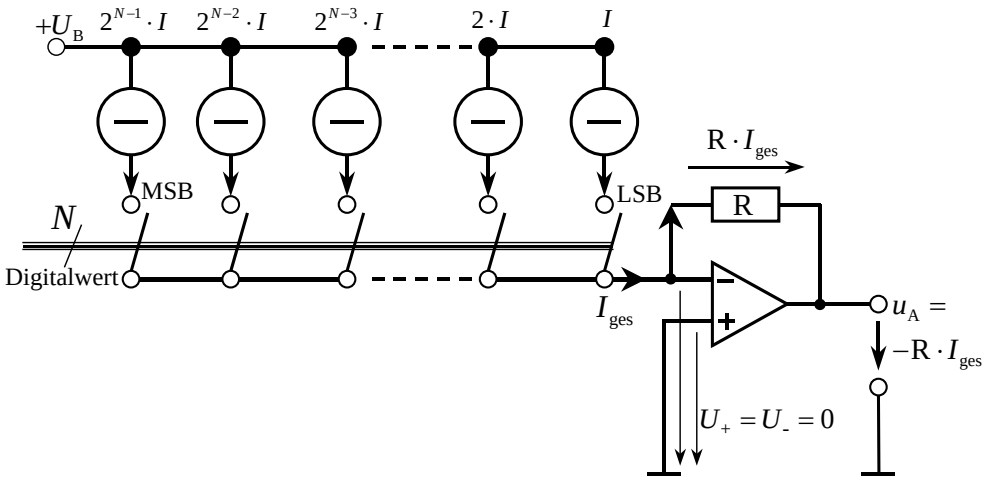
Gemäß Tabelle 3.2 füllen sich die Ergebnisregister in acht Taktschritten. Jetzt ist die Pipeline voll, so daß die Ausgangsstufen der Register $DREG_i$ (jeweils rechts) mit jedem folgenden Takt ein komplettes Wandelergebnis liefern. Im dargestellten Beispiel kommt natürlich achtmal der gleiche Digitalwert 10011001 heraus. Er entspricht der Spannung u_D= 5,9766V.

Tabelle 3.3: Bestimmung des gewandelten Digitalwertes zum Vergleich

5V	5/2	5/4	5/8	5/16	5/32	5/64	5/128	$\boldsymbol{U_D}$
1	0	0	1	1	0	0	1	
5	0	0	0,625	0,3125	0	0	0,0391	**5,9766**

Wie man sieht, liegt der Fehler mit 23,4mV in der Größenordnung von ½ LSB (1 LSB≙39,1mV).

3.2.5 Konventionelle Verfahren der D/A-Wandlung



Die Funktionsweise konventioneller D/A-Wandler ist generell einfacher zu verstehen als die der A/D-Wandler. Die technische Realisierung kann jedoch bei hohen Anforderungen an Geschwindigkeit und Genauigkeit sehr komplex und aufwendig werden.

Bild 3.14: Prinzip der D/A-Wandlung mit Stromquellen

Bild 3.14 verdeutlicht das Prinzip eines einfachen stromquellenbasierten Wandlers. Die Hauptbestandteile sind *N* Stromquellen, deren Ströme in Zweierpotenzen abgestuft sind. Nach Maßgabe eines *N* bit breiten Digitalwertes

$$d_N = \sum_{i=0}^{N-1} b_i \cdot 2^i \,, \quad \text{mit } b_i \in \{0,1\} \tag{3.26}$$

werden die eingezeichneten Schalter betätigt, d.h. geschlossen falls b_i=1 vorliegt und geöffnet für b_i=0. Alle Ströme, die über die Schalter kommen, fließen auf den –Eingang des Operationsverstärkers, so daß dessen Ausgangsspannung den Wert

$$-u_A = R \cdot I \cdot \left[\sum_{i=0}^{N-1} b_i \cdot 2^i \right] \tag{3.27}$$

annimmt.

3.2.5.1 Der R/2R-Wandler

Beim dem in Bild 3.15 dargestellten R/2R-Wandlerprinzip, das seinen Namen vom dem Widerstandsnetzwerk hat, in dem nur zwei verschiedene Werte vorkommen, die zudem exakt im Verhältnis 1:2 stehen, wird wiederum am virtuellen Nullpunkt eines Operationsverstärkers eine Stromsumme gebildet, so daß am Ausgang eine dazu proportionale Spannung u_A auftritt. Im Unterschied zu Bild 3.14 haben die durch die einzelnen Bits des Digitalwertes betätigten Schalter hier zwei Positionen, d.h. bei b_i=0 besteht eine Verbindung zur Masse, die den Strom aufnimmt, und bei b_i=1 fließt der Strom auf den virtuellen Nullpunkt des OP, so daß Spannungs- und Stromverhältnisse im Netzwerk stets unabhängig von den Schalterstellungen sind. Wie man aus Bild 3.15 erkennt, liefert das LSB z.B. den Strom

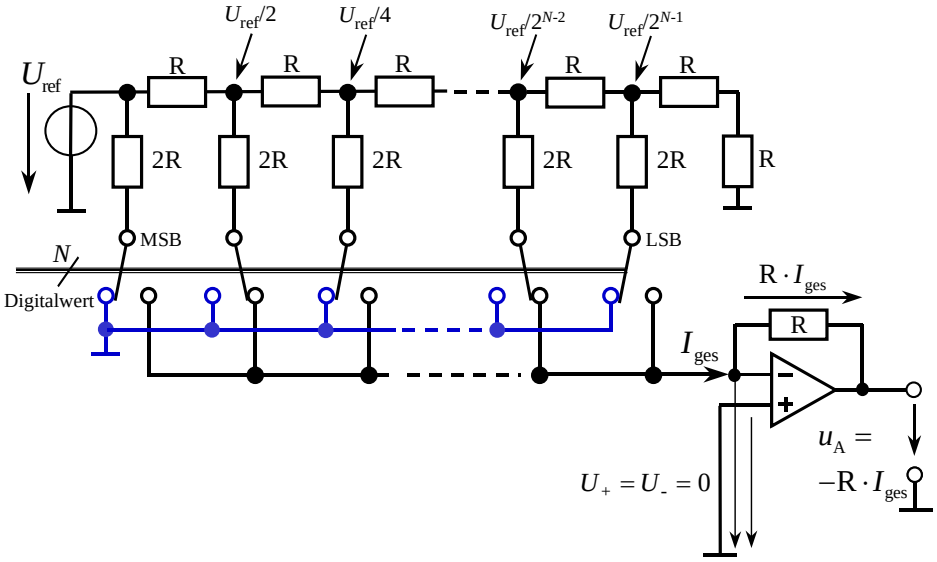


Bild 3.15: D/A-Wandler nach dem R/2R-Prinzip

$$I_{LSB} = \frac{U_{ref}}{2^{N-1} \cdot 2R} = \frac{U_{ref}}{2^N \cdot R}, \tag{3.28}$$

und das MSB

$$I_{MSB} = \frac{U_{ref}}{2R}. \tag{3.29}$$

Für einen N bit breiten Digitalwert nach (3.26) erhält man dann

$$-u_A = R \cdot \left[\sum_{i=0}^{N-1} \frac{U_{ref}}{2^N \cdot R} b_i \cdot 2^i \right] = \sum_{i=0}^{N-1} \frac{U_{ref}}{2^N} b_i \cdot 2^i . \tag{3.30}$$

3.2.5.2 Sigma-Delta-Prinzip zur A/D-Wandlung

Wenn es darum geht, hochauflösende Analog-Digitalwandler (mit über 20 bit) zu bauen, die kostengünstig und integrierbar sind, dann kommt eine Technik zum Einsatz, die Sigma-Delta-Prinzip genannt wird. Obwohl die Grundlagen schon 1962 bekannt waren, hatten Sigma-Delta-Wandler (ΣΔW) in der Vergangenheit kaum Bedeutung. Erst mit den Möglichkeiten der Hochintegration (VLSI-Technologie) konnten für das Sigma-Delta-Prinzip umfangreiche Anwendungsgebiete erschlossen werden. Denn eine entscheidende Anforderung an moderne A/D-Wandler ist die Kompatibilität mit hochintegrierter digitaler Signalverarbeitungshardware. Am günstigsten ist diese Anforderung zu erfüllen, wenn sowohl Analog- wie auch Digitalteil auf einem Stück Silizium monolithisch integriert werden können. Das gelingt mit dem Sigma-Delta-Wandler, weil er umfangreiche digitale Filterung benötigt, aber nur wenige analoge Komponenten, an die keine besonders hohen Anforderungen zu stellen sind. Rund 90% der Siliziumfläche eines Sigma-Delta-Wandlers beinhalten rein digitale Funktionen.

Die ständig wachsende Anwendung leistungsfähiger digitaler Signalprozessoren (DSP) hat den Einsatz von Sigma-Delta-Wandlern stark forciert, weil sie mit dem DSP monolithisch integrierbar sind. Durch die digitale Realisierung der Schlüsselkomponenten erreicht man hohe Auflösung bei sehr geringen Parameterstreuungen. Eine Nachbearbeitung, z.B. durch Laser-Trimmen – wie sie für Präzisionswandler in andern Technologien häufig benötigt wird – ist beim ΣΔ-Wandler nicht erforderlich.

Das Integrieren gewöhnlicher A/D-Wandler zusammen mit schnellen Digitalschaltungen erreicht bei etwa 12bit eine Grenze, weil Probleme der ***elektromagnetischen Verträglichkeit*** **(EMV)** zutage treten, die sich als schwer beherrschbar erweisen. Zudem wird hohe Präzision von der Referenzspannungsquelle, dem D/A-Wandler und dem Komparator gefordert. Die summierten Fehler müssen stets kleiner als eine halbe Amplitudenstufe (≡ ½ LSB) bleiben. Bei einem 10 bit-Wandler, der z.B. eine maximale Eingangsamplitude von 1V verarbeiten kann, entspricht das LSB ca. 1mV. Störspannungen in dieser Größenordnung entstehen sehr leicht, wenn in unmittelbarer Nähe digitale Signale mit steilen Flanken und einem Spannungshub von 3,3 oder gar 5V geführt werden.

Das Sigma-Delta-Prinzip umgeht all diese Probleme, dadurch daß eine hohe Überabtastung mit einer extrem groben Quantisierung kombiniert wird. In Bild 3.16 sind vorab die wesentlichen Funktionsblöcke dargestellt, die im weiteren detailliert behandelt werden. Man sieht, daß keine Referenzspannung benötigt wird. Der Komparator entscheidet nur über das Vorzeichen, d.h. er hat die Schwelle Null und liefert ein „1“-Bit, wenn das Eingangsignal positiv ist und ansonsten ein „0“-Bit.

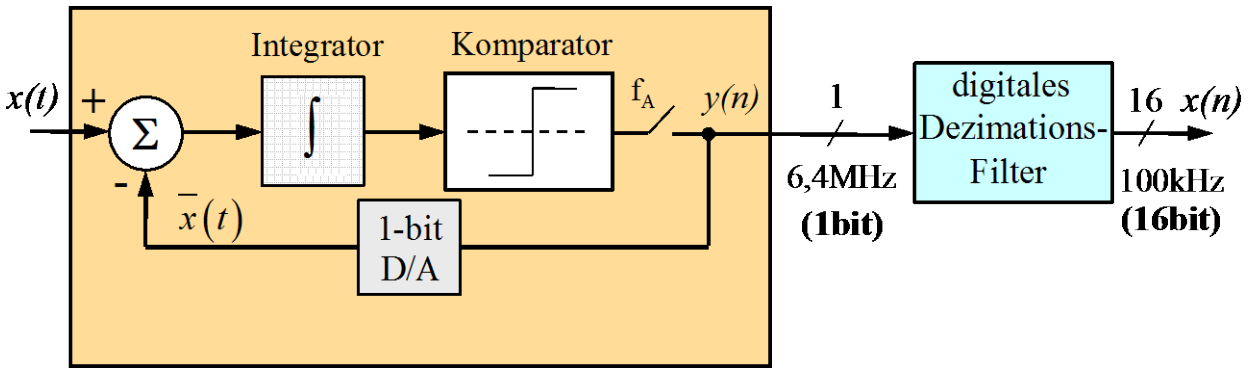


Bild 3.16: Einführungsbeispiel zum Sigma-Delta-Wandlerprinzip

Der D/A-Wandler in der Rückführung ist praktisch eine direkte Verbindung. Summierer und Integrator können zusammen in Form eines Operationsverstärker realisiert werden. Eine Analyse der links eingerahmten Schaltung – die weiter unten durchgeführt wird – zeigt, daß das Quantisie-

rungsrauschen, das wegen der extrem groben Stufung hier eine ganz erhebliche Leistung hat, zum einen eine spektrale Spreizung (durch Überabtastung) und zum anderen eine spezielle Formung (engl.: ***Noise Shaping***) erfährt. Diese beiden Effekte bewirken, daß der größte Teil der Rauschleistung zu sehr hohen Frequenzen hin verschoben und so das interessierende Frequenzband quasi rauschfrei gemacht wird. Dazu dient das im rechten Teil von Bild 3.16 eingezeichnete digitale Dezimations-Filter. Es erfüllt zwei Aufgaben: Die unerwünschten Rauschanteile werden weggefiltert und die Überabtastung wird schrittweise reduziert. Auf diese Weise erhält man aus einem schnellen, aber nur 1 bit breiten Datenstrom *y(n)* schließlich das gewünschte hochaufgelöste und verlangsamte Digitalsignal *x(n)*.

Die Anforderungen an den Anti-Aliasing-Tiefpaß in Bild 3.3 können durch Erhöhen der Abtastfrequenz über die Nyquistrate hinaus abgemildert werden. Neben der Unterdrückung von Aliasing stellt die Quantisierung bei gewöhnlichen hochauflösenden A/D-Wandlern ein nicht zu unterschätzendes Problem dar. Allein die Bereitstellung einer hinreichend exakten Referenz ist bei einem 16 bit-Wandler mit ±1V Eingangsspannungsbereich schon eine Herausforderung, da eine Quantisierungsstufe nur 30µV umfaßt. Auch die Linearitätsanforderungen an die analogen Komponenten machen bei der Herstellung sehr zu schaffen. In der Regel kommt man ohne nachträgliche Laser-Trimmung nicht aus, was sich entsprechend auf die Herstellungskosten niederschlägt.

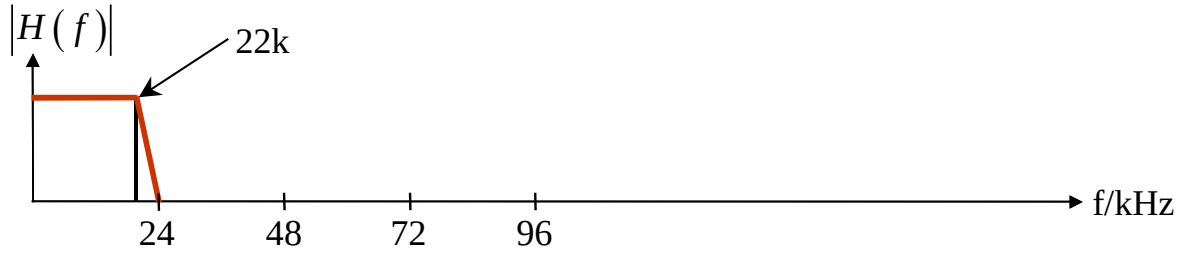
Grundsätzlich ist jeder Quantisierungsprozeß fehlerbehaftet. Unter Annahme einer Gleichverteilung des Fehlers innerhalb einer Quantisierungsstufe wurde mit (3.20) die Leistung des Quantisierungsrauschens berechnet, wobei das Quantisierungsintervall Δ sich nach (3.17) aus der maximalen Eingangsspannungsamplitude ergibt. Geht man davon aus, daß das Quantisierungsrauschen im interessierenden Frequenzbereich weiß ist, d.h. spektral gleichmäßig in einem Intervall der Breite $f_A=2\cdot f_g$ verteilt ist, dann ergibt sich die spektrale Leistungsdichte

$$n_q = \frac{\Delta^2}{12 \cdot f_A} = \frac{\Delta^2}{24 \cdot f_g} \tag{3.31}$$

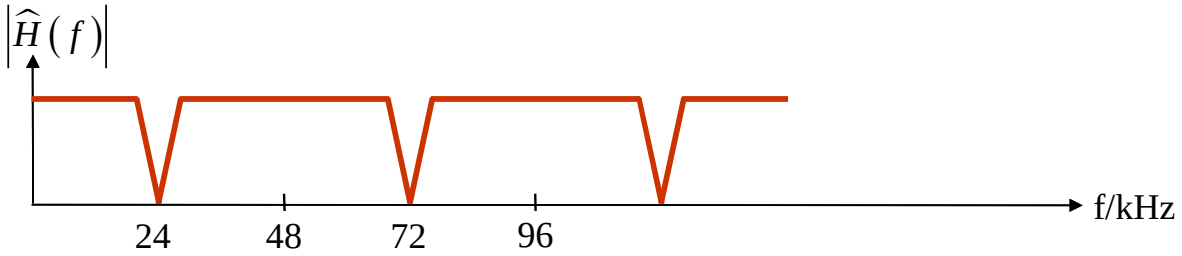
für das Quantisierungsrauschen. Mit diesen Betrachtungen können jetzt die Begriffe ***Oversampling und Dezimation*** erläutert werden. Während ein gewöhnlicher A/D-Wandler in einem einzigen Abtastintervall die Quantisierung des Abtastwertes mit der vollen Genauigkeit von *Z* bit $\equiv 2^Z$ Stufen vornimmt, verwendet ein Wandler mit ***Oversampling*** eine Vielzahl aufeinander folgender grob quantisierter Daten, um in einem ***Dezimationsprozeß***, der rein digital erfolgt, einen schrittweise genauer werdenden Schätzwert des analogen Eingangssignals zu erhalten. Die Geschwindigkeit, mit der der Schätzwert verfügbar wird, nähert sich in der Regel der Nyquistrate eines gewöhnlichen Wandlers.

Worin liegen die wesentlichen Vorteile des Oversampling?

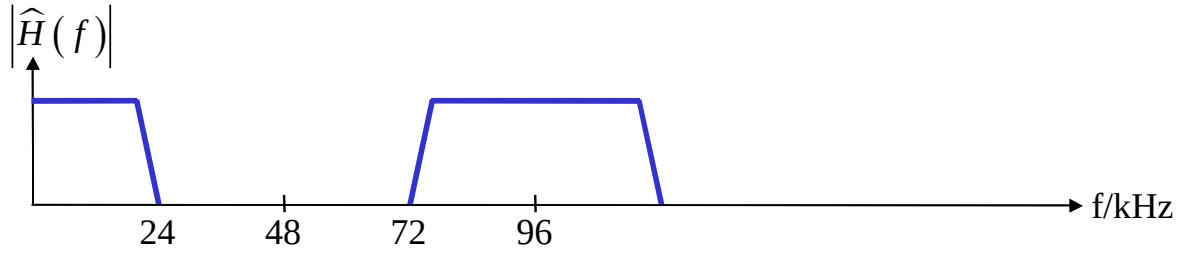
Läßt man die Quantisierung zunächst außer acht, dann ergeben sich bereits unmittelbare Vorteile des ***Oversampling*** für die Realisierung des Anti-Aliasingfilters. Beim Wandeln mit Nyquistrate werden dem Signal bekanntlich Abtastwerte mit einer Rate entnommen, die mindestens dem Doppelte der höchsten vorkommenden Frequenz entspricht. So erlaubt es beispielsweise eine Abtastrate von 48kHz, Signale mit Frequenzen bis etwa 22 kHz ohne Aliasing zu verarbeiten. Das Filter muß zumindest im interessierenden Frequenzband (ca. 20 kHz für Audioanwendungen) eine flache Übertragungsfunktion und einen linearen Phasenverlauf aufweisen. Um Aliasing sicher zu verhindern, müssen für eine Auflösung von 16 bit - bei einer Abtastrate f_A=48 kHz - alle Frequenzen über 24 kHz um mindestens 96 dB gedämpft sein. Diese Anforderung ist mit einem analogen Tiefpaßfilter sehr schwer zu erfüllen. Bild 3.17 a) zeigt die benötigte Übertragungsfunktion eines analogen Anti-Aliasingfilters bei Abtastung mit der Nyquistrate, während in b) das Spektrum des digitalisierten Signals dargestellt ist.



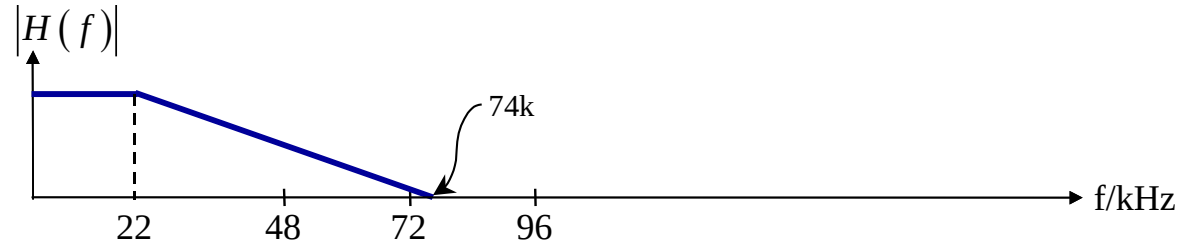
(a) Anti-Aliasingfilter für Abtastung mit Nyquist-Rate



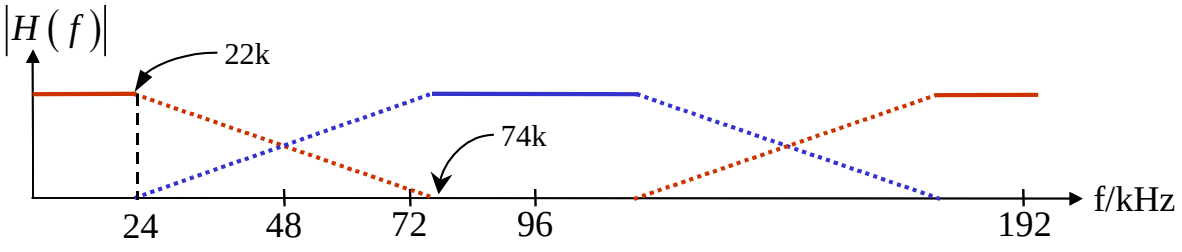
(b) Spektrum nach Abtastung mit f_A=48kHz



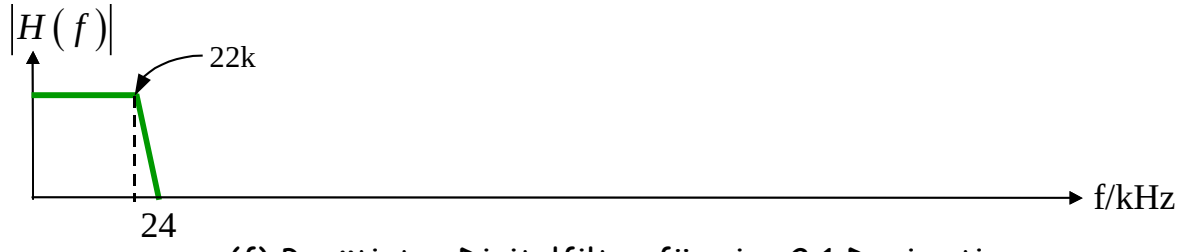
(c) Spektrum nach Abtastung mit f_A=96kHz



(d) Anti-Aliasingfilter bei zweifacher Überabtastung



(e) Spektrum nach zweifacher Überabtastung mit f_A=96kHz



(f) Benötigtes Digitalfilter für eine 2:1 Dezimation

Bild 3.17: Vergleich zwischen Abtastung mit Nyquistrate und zweifacher Überabtastung

Wird nun die Abtastfrequenz verdoppelt, d.h. f_A=96 kHz, dann muß das Anti-Aliasingfilter nur noch Signale über 74 kHz „beseitigen“, und lediglich bis 22 kHz einen flachen Verlauf der Übertragungsfunktion haben - s. Bild 3.17 c). Die Realisierung dieses Filters ist erheblich einfacher, weil eine Übergangsbereich von 52 kHz (22...74 kHz) zum Erreichen der 96 dB Dämpfung zur Verfügung steht, d.h. es genügt eine erheblich flachere Filterflanke.

Da jedoch die endgültige Abtastrate wieder 48 kHz sein soll, wird ein so genanntes ***Dezimationsfilter*** benötigt, das im Gegensatz zum Anti-Aliasingfilter rein digital realisiert wird. Bild 3.17 d) veranschaulicht die Anforderungen die bezüglich des Frequenzgangs bei f_A=96 kHz an das analoge Anti-Aliasingfilter zu stellen sind und unter e) ist das Spektrum des abgetasteten Signals dargestellt. Bild 3.17 f) zeigt schließlich den Frequenzgang, den ein digitales Dezimationsfilters haben muß, damit eine Reduktion der Abtastrate auf f_A=48 kHz erfolgen kann.

Über die zweifache Überabtastung hinaus, die anhand von Bild 3.17 betrachtet wurde, kann eine weitere Erhöhung der Abtastrate um einen Faktor $n \gg 2$ weitere Vorteile bringen. Bild 3.18 a) zeigt, daß sich die Anforderungen an einen Anti-Aliasing-Tiefpaß sehr stark vermindern. Eine hohe Dämpfung muß erst oberhalb einer Frequenz f_A-f_g erreicht werden. In Bild 3.18 b) ist das Spektrum des gesamten Quantisierungsrauschens und (schraffiert) der Bereich von -f_g...f_g des so genannten Basisbandrauschens, das nach dem Dezimationsprozeß verbleibt, dargestellt.

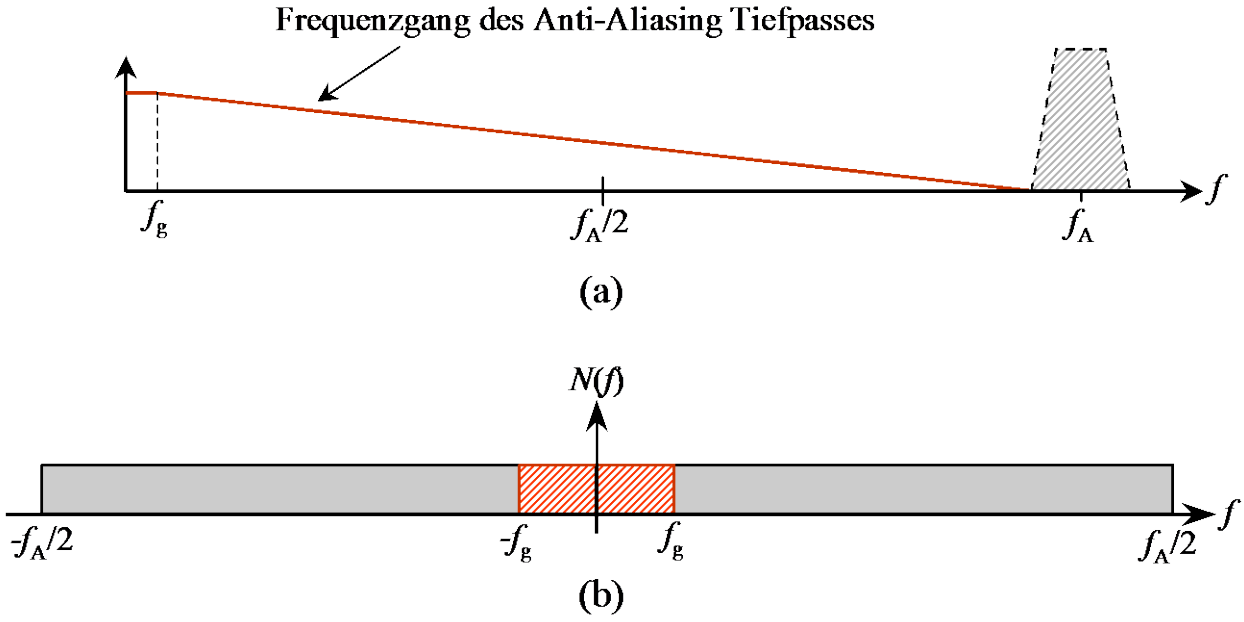


Bild 3.18: Die Auswirkungen von sehr starker Überabtastung

Hinsichtlich des Quantisierungsrauschens wirkt ***Oversampling*** wie ein Bandspreizverfahren (engl.: Spread Spectrum Technique). Wie Bild 3.18 b) zeigt, wird die Rauschleistung dabei über ein Frequenzband der Breite $f_A = n \cdot 2 f_g$ verteilt, so daß sich mit (3.20) und (3.31) die Rauschleistungsdichte

$$n_q(f) = \frac{\Delta^2}{12 \cdot f_A} = \frac{\Delta^2}{24 \cdot n \cdot f_g}, \quad \text{für } |f| \le \frac{f_A}{2} = \frac{n \cdot f_g}{2} \text{ und sonst 0} \tag{3.32}$$

ergibt. Das Nutzband, das später digital zu verarbeiten ist, wird durch digitales Filtern und Dezimation auf die Bandbreite $2f_g$ reduziert. Damit ist auch eine Reduktion der Rauschleistung in diesem Band auf

$$N_n = \int_{-f_g}^{f_g} n_q(f)\,df = \frac{\Delta^2 \cdot 2 f_g}{24 \cdot n \cdot f_g} = \frac{\Delta^2}{12 \cdot n} \tag{3.33}$$

verbunden. Im Vergleich zum gewöhnlichen A/D-Wandler mit der Abtastrate $f_A=2f_g$ bietet n-fache Überabtastung somit eine Rauschleistungsreduktion um den Faktor

$$\frac{\Delta^2 \cdot 12 \cdot n}{12 \cdot \Delta^2} = n \tag{3.34}$$

Die entscheidenden Vorteile ***hoher Überabtastung*** sind also

- geringe Anforderungen an einen Anti-Aliasing-Tiefpaß vor dem Wandler
- Reduktion des Quantisierungsrauschens

Aus letzterem läßt sich anhand von (3.24) und (3.25) ableiten, daß man den Dynamikbereich (bzw. die Auflösung) eines A/D-Wandlers mittels Überabtastung erweitern kann, denn bei n-facher Überabtastung folgt für das Signal-Störverhältnis mit (3.24) und (3.34)

$$\left[\frac{S_s}{N_q}\right]_n = n \cdot \frac{3}{2} \cdot 2^{2Z}, \tag{3.35}$$

bzw. in der logarithmischen Darstellung

$$\frac{a}{\mathrm{dB}} = 1{,}76 + 6{,}02 \cdot Z + 10 \cdot \lg(n) \tag{3.36}$$

Wie man sieht, ergibt eine vierfache Überabtastung genau ein Bit mehr, da 10lg(4)=6,02 ist, d.h. man könnte in (3.36) den letzten Term weglassen und Z um Eins erhöhen. Um 2 bit zu gewinnen, muß bereits eine 16-fache Überabtastung erfolgen.

Im weiteren führen vertiefte Betrachtungen der Überabtastung in Verbindung mit der Deltamodulation auf das bereits in Bild 3.16 angedeutete Sigma-Delta-Wandlerkonzept. Der ΣΔW ermöglicht einen sehr flexiblen Austausch von Geschwindigkeit gegen Auflösung. Entsprechend Bild 3.16 liegt am Ausgang des ersten Wandlerabschnitts zunächst ein lediglich ein ***1bit*** breiter Datenstrom vor, der allerdings eine hohe Rate aufweist. Das folgende Dezimationsfilter ist dann für den Austausch von Geschwindigkeit gegen Auflösung zuständig. Zur quantitativen Analyse des ersten Funktionsblocks in Bild 3.16, der aus dem Analogsignal $x(t)$ die Bitfolge $y(n)$ erzeugt, wird zunächst die Deltamodulation betrachtet.

3.2.5.2.1 Die Deltamodulation

Ausgangspunkt der Entwicklung von Sigma-Delta-Wandlern war die Deltamodulation. Das Prinzip der Deltamodulation ist die Quantisierung der Änderung eines analogen Eingangssignals von Abtastwert zu Abtastwert. Also nicht der Absolutwert des Eingangssignals wird jeweils komplett in ein äquivalentes Digitalsignal umgesetzt, sondern nur die - normalerweise sehr kleine als ***Delta*** bezeichnete - Änderung des Analogsignals von Abtastzeitpunkt zu Abtastzeitpunkt. Bild 3.19 a) zeigt die Blockschaltung eines Deltamodulators und Beispiele für typische Ein- und Ausgangssignale. Der Deltamodulator besteht aus einem 1 bit-(*Vorzeichen*)-Quantisierer, der im Takt der Abtastrate $f_A=1/T$ das digitale Ausgangssignal $y(t)$ liefert. $y(t)$ wird einem Integrator zugeführt, dessen Ausgang einen Schätzwert $\bar{x}(t)$ für das analoge Eingangssignal $x(t)$ liefert. Der Summierer Σ liefert das Differenzsignal $x(t)-\bar{x}(t)$ an den Quantisierer, der bei $x(t)-\bar{x}(t)>0$ ein „H“-Bit und

sonst ein „L“-Bit in den Datenstrom $y(t)$ einfügt. Die Signalbilder zeigen leicht nachvollziehbar die Funktion. An flachen Stellen des Eingangssignals weist $y(t)$ häufige Bitwechsel auf (Mittelwert ≈ 0). Bei steilen Flanken von $x(t)$ bleibt $y(t)$ dagegen über mehrere Abtasttakte gleich. Die Demodulation eines deltamodulierten Signals ist denkbar einfach wie Bild 3.19 b) zeigt: $y(t)$ wird integriert, so daß wie im Modulator $\bar{x}(t)$ entsteht, das den Verlauf des ursprünglichen Analogsignals $x(t)$ approximiert. Durch Tiefpaßfilterung wird schließlich $x(t)$ aus $\bar{x}(t)$ zurückgewonnen.

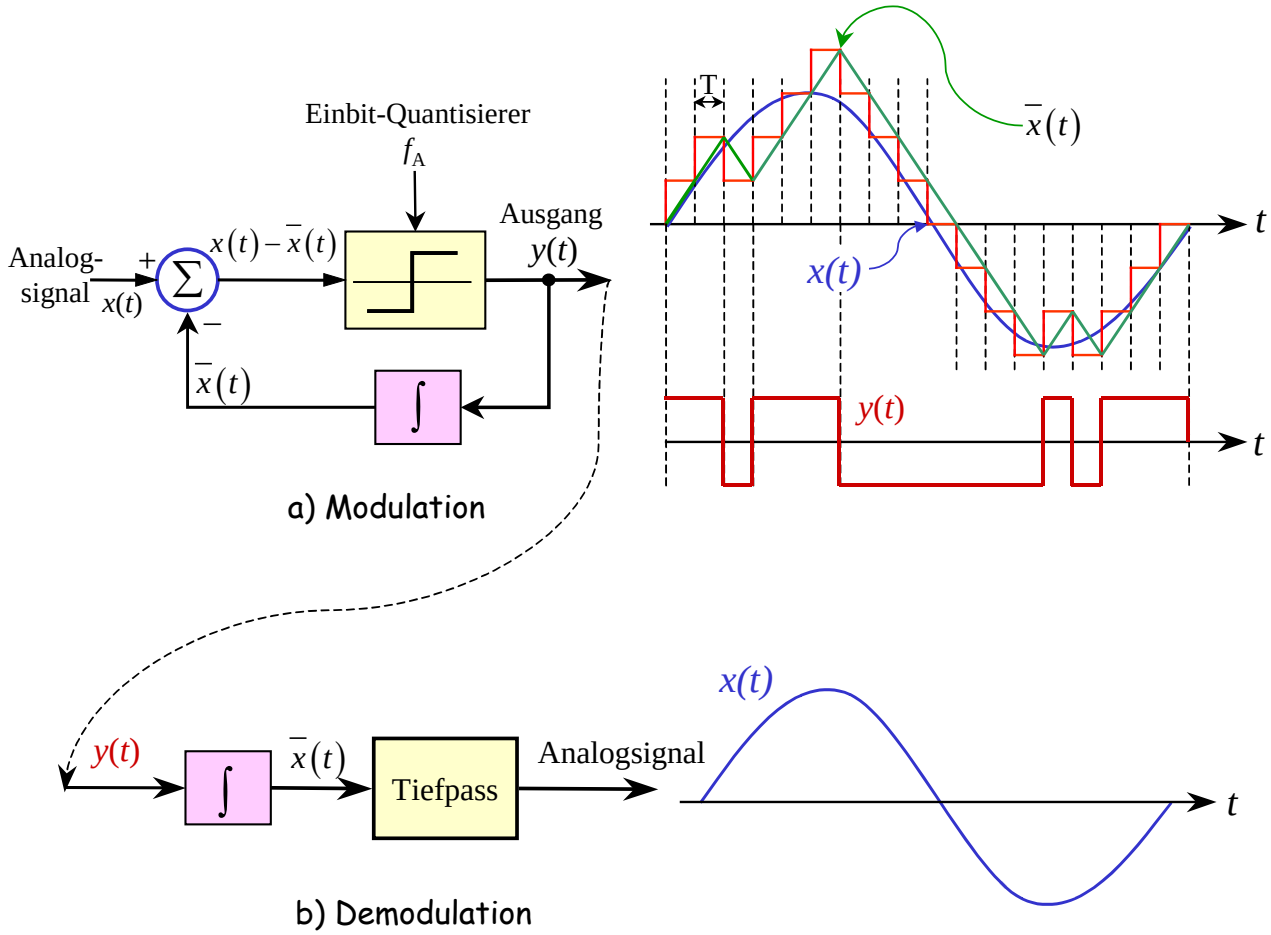


Bild 3.19: Deltamodulation und -demodulation

Modifikation der Sigma-Delta-Modulation zum Aufbau von A/D-Wandlern

Bei der Deltamodulation und -demodulation werden u.a. zwei Integratoren benötigt – wie aus Bild 3.19 hervorgeht.

Weil Integrieren eine lineare Operation ist, kann der Integrator vom Eingang des Deltademodulators ohne Veränderung des gesamten Systemverhaltens auch an den Eingang des Modulators verschoben werden – s. Bild 3.20. Wegen der Linearität von Integration und Summation ist jetzt ein Vereinfachungsschritt möglich, der in Bild 3.21 gezeigt ist. Das Ergebnis ist praktisch schon der Grundaufbau des Sigma-Delta-Wandlers. In einem letzten Modifikationsschritt wird der Kanal weggelassen und eine direkte Verbindung zwischen Modulator und Demodulator hergestellt – s. Bild 3.22. Da nun ein rein digitales Signal vorliegt, kann der analoge Tiefpaß durch ein digitales Filter ersetzt werden, das schrittweise auch eine Dezimation vornimmt. Die Rückführung zum Summationsknoten ist ein D/A-Wandler mit 1 bit Auflösung, der lediglich die Aufgabe hat, den Logikpegel des digitalen Ausgangssignals an den Pegel des Eingangssignals anzupassen.

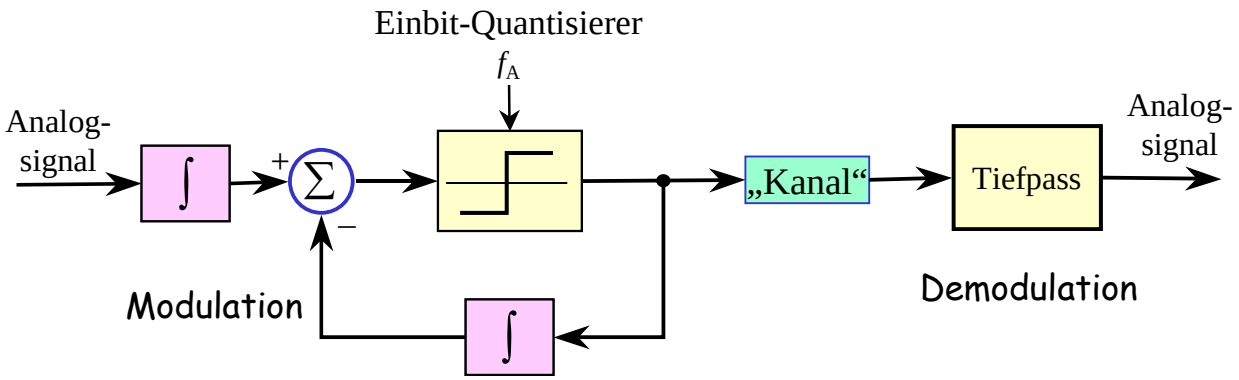


Bild 3.20: Zusammenfassung von Modulation und Demodulation

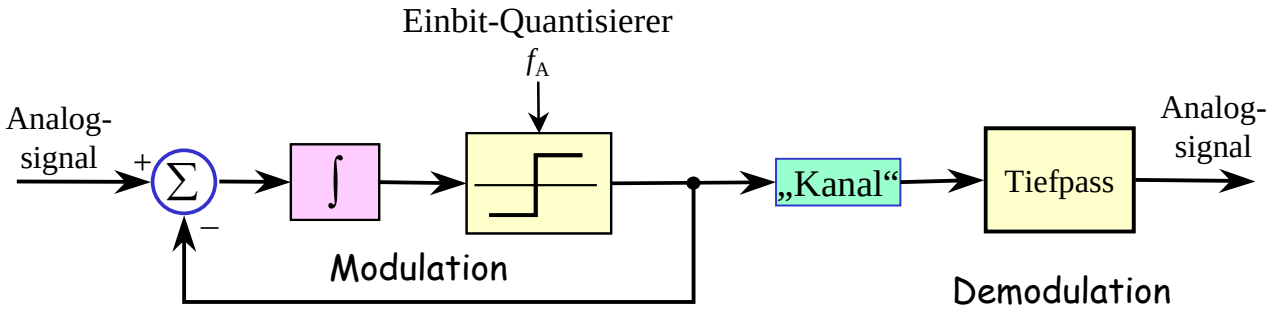


Bild 3.21: Reduktion auf einen Integrator

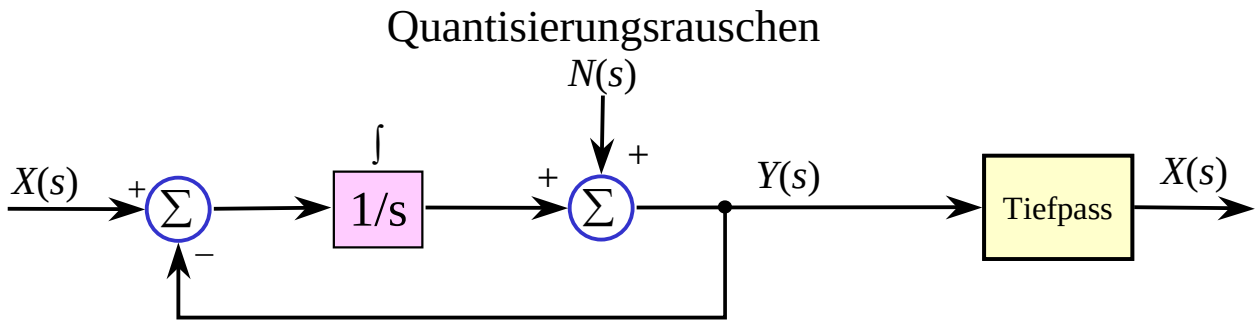


Bild 3.22: Aufbereitung zur Analyse mittels Laplacetransformation

Hinter dem Summationsknoten führt der ΣΔW eine Integration der Signaldifferenz $x(t)-\bar{x}(t)$, die dem Quantisierungsfehler entspricht, durch und quantisiert das Integrationsergebnis. Obwohl der Fehler wegen der groben 1 bit-Quantisierung ziemlich groß ist sind trotzdem hochaufgelöste Wandelergebnisse aus dem digitalen Datenstrom zu erhalten. Dazu ist es erforderlich, über zahlreiche Abtastperioden zu mitteln. Diese Aufgabe erfüllt das digitale Dezimationsfilter.

Eine Analyse des Verhaltens eines ΣΔW mit Hilfe der Laplacetransformation kann anhand von Bild 3.22 durchgeführt werden.

Die Übertragungsfunktion *H(s)* für das Nutzsignal ergibt sich für den Fall, daß das Quantisierungsrauschen $N(s)=0$ ist zu

$$H(s)=\frac{Y(s)}{X(s)}=\frac{1}{1+s}, \tag{3.37}$$

d.h. es liegt Tiefpaßverhalten vor. Für $X(s)=0$ kann die Rauschübertragungsfunktion $R(s)$ ermittelt werden. Es ist

$$R(s)=\frac{Y(s)}{N(s)}=\frac{s}{1+s}. \tag{3.38}$$

Das Rauschen wird also durch Hochpaßfilterung beeinflußt. Die Resultate sind anschaulich in Bild 3.23 zusammengefaßt. Im rechten Teil des Bildes erkennt man das, was unter dem Begriff 'Noise Shaping' zu verstehen ist. Man sieht, daß der ΣΔW die Rauschleistung zu höheren Frequenzen hin verschiebt und sie gleichzeitig bei tiefen Frequenzen sehr stark absenkt.

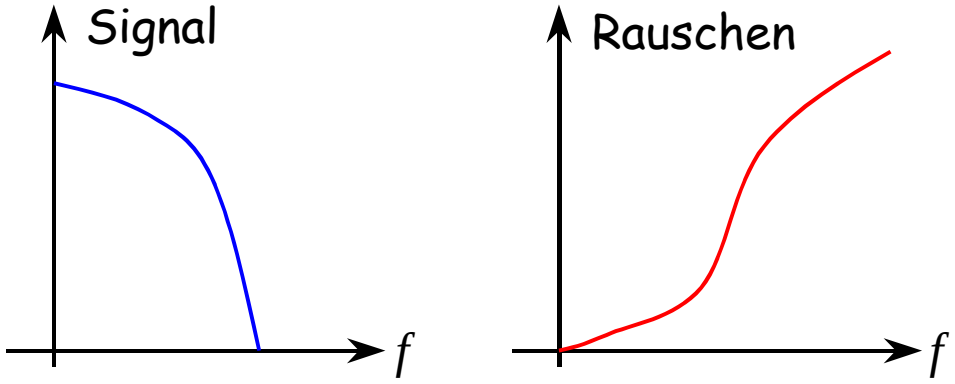


Bild 3.23: Anschauliche Darstellung der Wirkung eines ΣΔ-Loops

Die bisherige Analyse beschränkt sich auf einen ΣΔ-Loop 1. Ordnung. Weiter unten werden Loops mit höherer Ordnung (bis zur 3.) betrachtet, die sich im wesentlichen vom Loop 1. Ordnung dadurch unterscheiden, daß der Noise Shaping Effekt weiter verstärkt wird, d.h. das Rauschen gemäß Bild 3.23 wird im unteren (interessierenden) Frequenzbereich noch mehr abgesenkt und erscheint dafür verstärkt im oberen Bereich, der schon weit außerhalb der Nutzbandbreite liegt. Bei Betrachtung von Loops höherer Ordnung ist die Anwendung der z-Transformation von Vorteil. Deshalb wird diese zunächst auf den bekannten Loop 1. Ordnung angewendet.

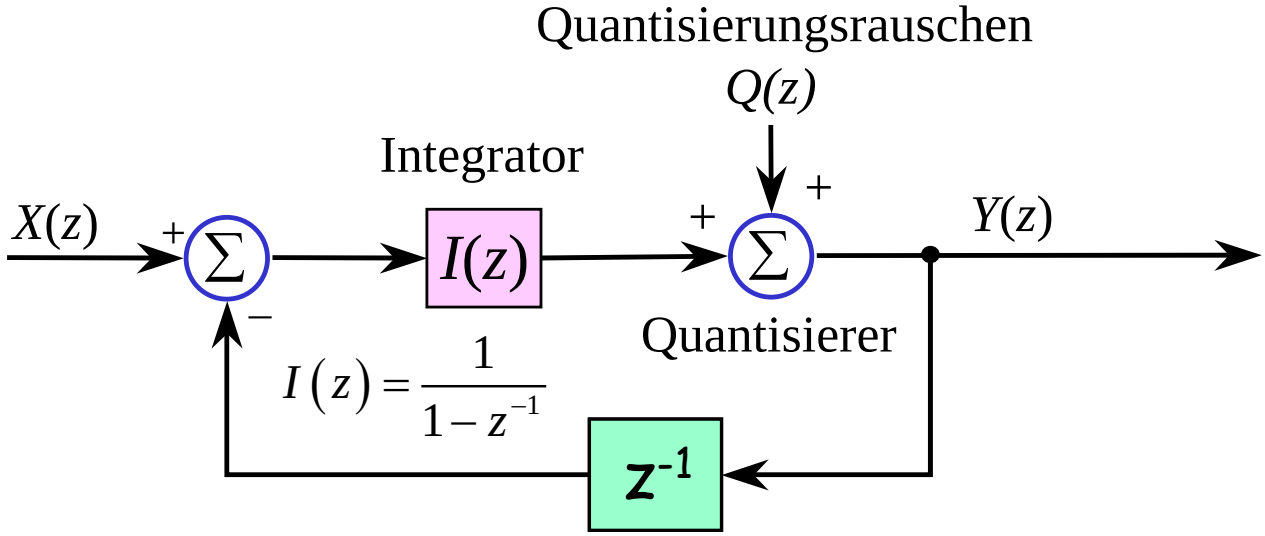


Bild 3.24: Darstellung eines ΣΔ-Loops 1. Ordnung für die Analyse mittels z-Transformation

Mit den in Bild 3.24 eingeführten Bezeichnungen und der Übertragungsfunktion *I*(z) des Integrators erhält man für die z-Transformierte des Ausgangsignals

$$Y(z)=Q(z)+I(z)\cdot\left[X(z)-z^{-1}\cdot Y(z)\right]. \tag{3.39}$$

Diese Gleichung kann einfach nach *Y*(*z*) aufgelöst werden und es ergibt sich

$$Y(z) = X(z) \cdot \frac{I(z)}{1 + I(z) \cdot z^{-1}} + Q(z) \cdot \frac{1}{1 + I(z) \cdot z^{-1}}. \tag{3.40}$$

Da die z-Transformierte eines idealer Integrator durch

$$I(z) = \frac{1}{1 - z^{-1}} \tag{3.41}$$

gegeben ist, folgt für den Ausgang ΣΔ-Loops 1. Ordnung

$$Y(z) = X(z) + \left(1 - z^{-1}\right) \cdot Q(z) \tag{3.42}$$

Das Quantisierungsrauschen wird also einer Differentiation ($1-z^{-1}$) unterzogen, die dafür sorgt, daß die Rauschleistung zu höheren Frequenzen hin verschoben wird. Unter der Voraussetzung, daß eine hohe Überabtastung des analogen Nutzsignals $x(t)$ vorliegt, kann das zu hohen Frequenzen verschobene Quantisierungsrauschen jetzt mittels digitaler Tiefpaßfilterung beseitigt werden, ohne daß dabei das Nutzfrequenzband beeinflußt wird. Diese Filterung ist eine entscheidende Vorbereitung für den Dezimationsprozeß. Nach Abschluß der Dezimation verbleibt am Ausgang nur noch der Frequenzbereich von $0...f_g$, so daß nunmehr ein Vergleich mit konventionellen Wandlern ohne Überabtastung möglich ist. Obwohl bereits für einen ΣΔ-Loop 1. Ordnung das Quantisierungsrauschen im Nutzband erheblich geringer ist als beim Wandler mit Nyquistrate werden die für eine 16 bit Auflösung benötigten 96dB noch nicht erreicht. Deshalb ist es für den Aufbau von ΣΔ-Wandlern für Audioanwendungen erforderlich, ΣΔ-Loops höherer Ordnung einzusetzen. Mit einem Loop 3. Ordnung – der im weiteren analysiert wird - erreicht man schließlich den gewünschten Störabstand von 96dB.

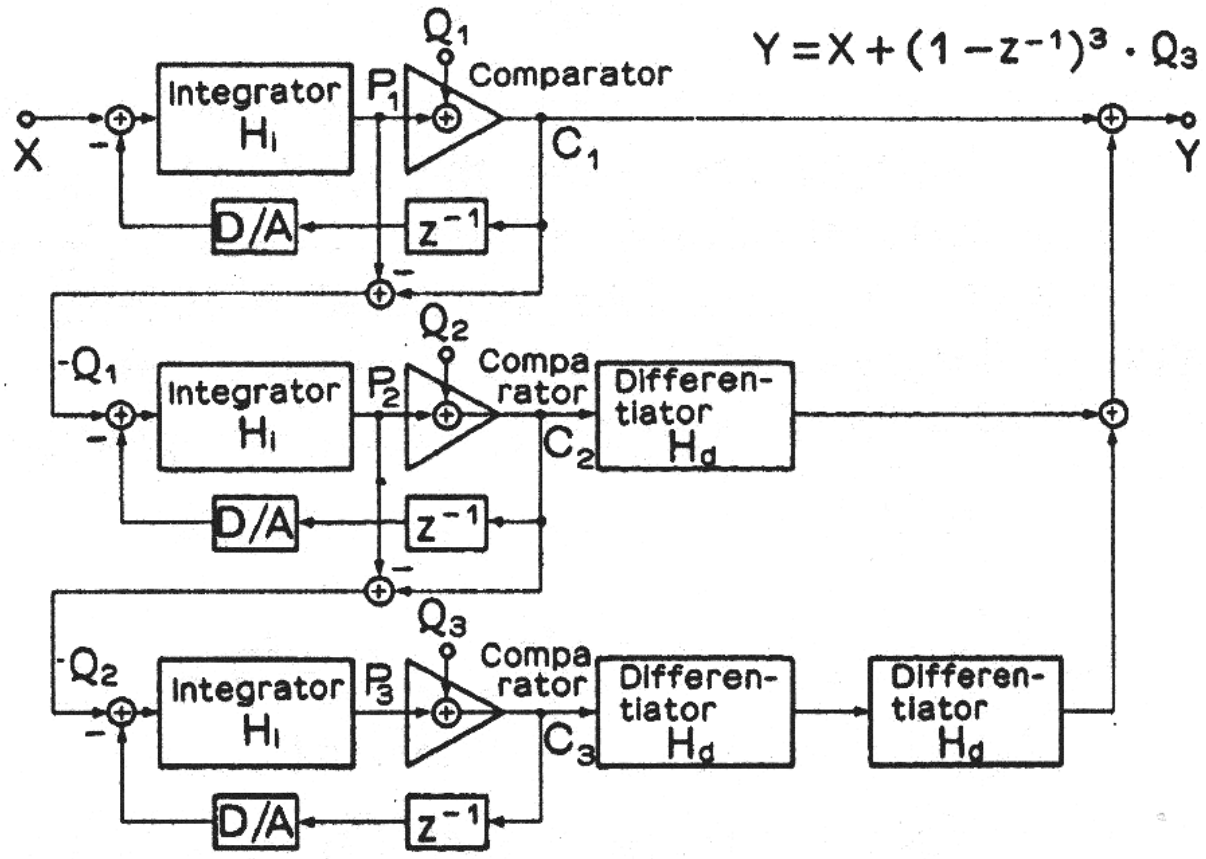


Bild 3.25: Darstellung eines ΣΔ-Loops 3. Ordnung für die Analyse mittels z-Transformation

Für das Ausgangssignal C_1 des ΣΔ-Loop 1. erster Ordnung in Bild 3.25 erhält man

$$C_1(z) = X(z) + \left(1 - z^{-1}\right) \cdot Q_1(z) \tag{3.43}$$

Wenn mehrere ΣΔ-Loops erster Ordnung kaskadiert werden, um eine höhere Ordnung zu erhalten, dann ist das Signal, das jeweils zu der nächstfolgenden Schleife geführt wird, stets der „Störterm“ aus der gerade betrachteten Schleife. Dieser Störterm $-Q_1$ ist die Differenz zwischen Integrator- und Quantisiererausgangssignal, d.h. $P_1 - C_1$ in Bild 3.25. Im 2. Loop sind die Eingangssignale $-Q_1$ und Q_2 und man erhält am Ausgang

$$C_2(z) = -Q_1(z) + \left(1 - z^{-1}\right) \cdot Q_2(z), \tag{3.44}$$

und entsprechend am Ausgang des dritten ΣΔ-Loops:

$$C_3(z) = -Q_2(z) + \left(1 - z^{-1}\right) \cdot Q_3(z). \tag{3.45}$$

Aus(3.43) ergibt sich

$$Q_1(z) = \frac{C_1(z) - X(z)}{1 - z^{-1}}. \tag{3.46}$$

Damit erhält man aus (3.44):

$$C_2(z) = \frac{X(z) - C_1(z)}{1 - z^{-1}} + \left(1 - z^{-1}\right) Q_2(z) \tag{3.47}$$

Gemäß Bild 3.25 ist das Ausgangssignal des ΣΔ-Loops 2. Ordnung jetzt:

$$Y_2(z) = C_1(z) + \left(1 - z^{-1}\right) \cdot C_2(z) = X(z) + \left(1 - z^{-1}\right)^2 \cdot Q_2(z). \tag{3.48}$$

In ähnlicher Weise kann $Q_2(z)$ aus (3.47) unter Verwendung von (3.48) bestimmt werden:

$$Q_2(z) = \frac{C_2(z) + Q_1(z)}{\left(1 - z^{-1}\right)} = \frac{C_2(z)}{\left(1 - z^{-1}\right)} + \frac{C_1(z) - X(z)}{\left(1 - z^{-1}\right)^2} \tag{3.49}$$

Damit ergibt sich

$$C_3(z) = \frac{-C_2(z)}{\left(1 - z^{-1}\right)} + \frac{X(z) - C_1(z)}{\left(1 - z^{-1}\right)^2} + \left(1 - z^{-1}\right) \cdot Q_3(z). \tag{3.50}$$

Entsprechend der Struktur nach Bild 3.25 erhält man schließlich mit (3.50) das Ergebnis für den kompletten ΣΔ-Loop dritter Ordnung:

$$\begin{aligned} Y(z) &= C_1(z) + \left(1 - z^{-1}\right) \cdot C_2(z) + \left(1 - z^{-1}\right)^2 \cdot C_3(z) = \\ &= C_1(z) + \left(1 - z^{-1}\right) C_2(z) - \left(1 - z^{-1}\right) C_2(z) \\ &\quad + X(z) - C_1(z) + \left(1 - z^{-1}\right)^3 Q_3(z) = \\ &= X(z) + \left(1 - z^{-1}\right)^3 Q_3(z) \end{aligned} \tag{3.51}$$

Im Vergleich zu einem Loop 1. Ordnung erfährt das Quantisierungsrauschen aufgrund der dreifachen Differentiation eine starke Verschiebung zu höheren Frequenzen hin. Interessant ist auch, daß nur noch das Quantisierungsrauschen Q_3 ins Ausgangssignal eingeht.

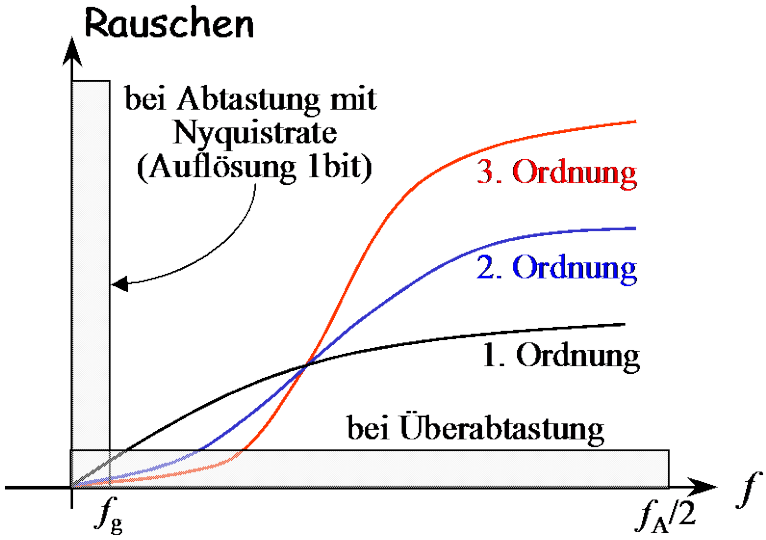
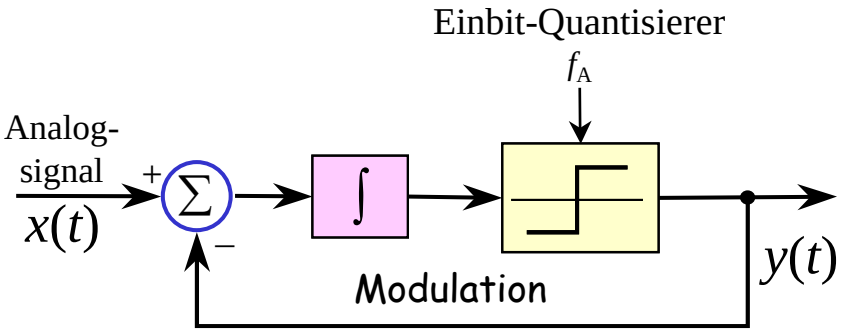


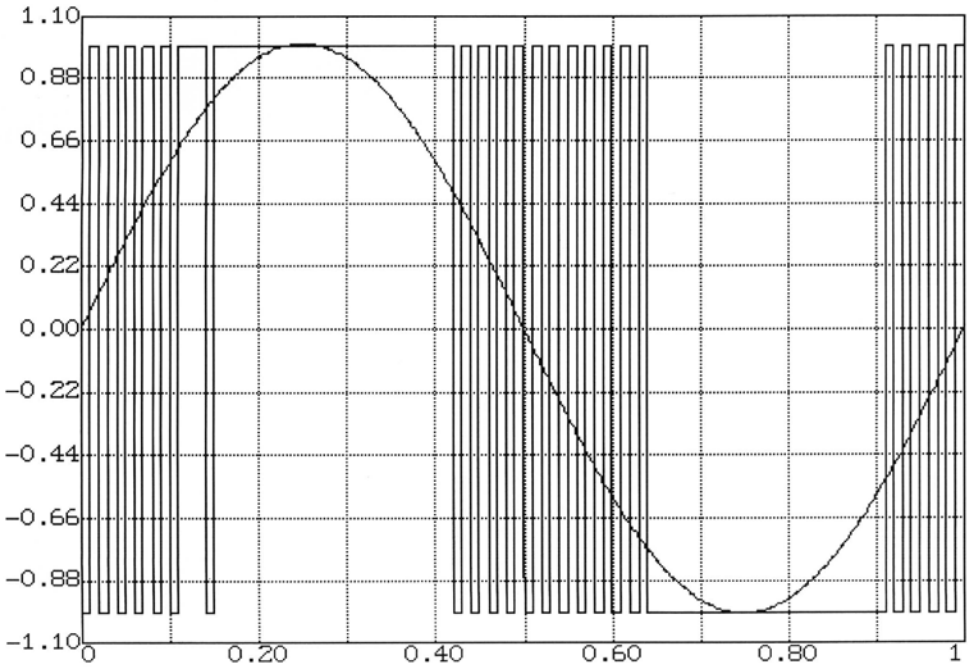
Bild 3.26 verdeutlich qualitativ die Wirkung der ΣΔ-Loops verschiedener Ordnung. Wie man sieht, steigt die Unterdrückung des Rauschens im Nutzband (bis f_g) ganz erheblich, wenn man einen Loop dritter Ordnung anstelle eines Loops 1. Ordnung verwendet. Das Bild macht auch klar, daß dieser Effekt allein durch Oversampling praktisch nicht zu erreichen ist. Es wären enorm hohe Abtastraten erforderlich, so daß rasch die Geschwindigkeitsgrenzen der Hardware erreicht würden.

Bild 3.26: Wirkung von ΣΔ-Loops höherer Ordnung im Vergleich mit Überabtastung



Abschließend verdeutlicht Bild 3.28 an einem Signalbeispiel im Zeitbereich die Arbeitsweise eines ΣΔ-Loops 1. Ordnung, der in Bild 3.27 nochmals mit dem analogen Eingangssignal $x(t)$ und dem digitalen Ausgangsignal $y(t)$ dargestellt ist.

Bild 3.27: ΣΔ-Loop 1. Ordnung



$x(t)$ sei die in Bild 3.28 dargestellte Sinuskurve. Der Modulator führt sowohl das Abtasten als auch das Quantisieren durch. Für die Dauer einer Taktperiode ist das rechteckförmige Ausgangssignal $y(t)$, das zum Summenpunkt zurückgeführt wird, entweder +1 oder −1.

Bild 3.28: Signalverlauf am Eingang und Ausgang eines ΣΔ-Loops 1. Ordnung

Wenn $x(t)$ den positiven Maximalwert annimmt, ist $y(t)$ während der meisten Taktperioden am oberen (positiven) Anschlag, d.h. +1. Entsprechendes gilt für den negativen Maximalwert. In beiden Fällen folgt $y(t)$ quasi dem Eingangssignal. Wenn sich $x(t)$ rasch ändert, d.h. große Steigung aufweist, variiert $y(t)$ rasch zwischen ±1, so daß der Mittelwert ungefähr Null ist.

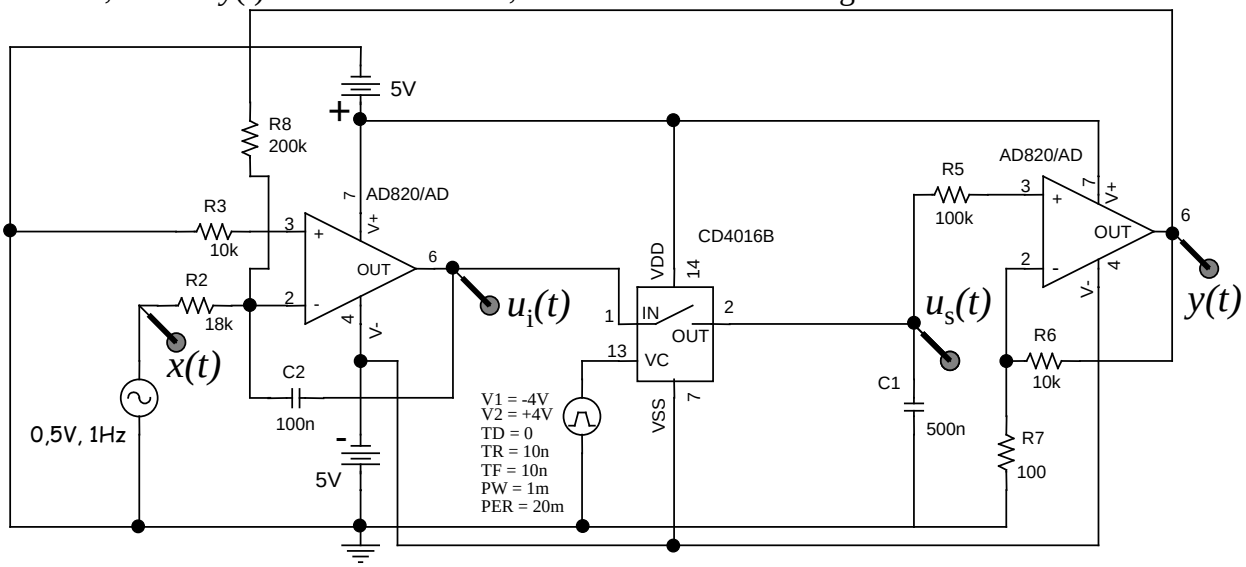


Bild 3.29: Schaltung zur Simulation des Zeitverhaltens eines ΣΔ-Loops 1. Ordnung mit PSPICE

Die Schaltung in Bild 3.29 stellt die technische Realisierung eines ΣΔ-Loops 1. Ordnung nach Bild 3.27 dar. Der erste Operationsverstärker bildet den Summenpunkt und führt gleichzeitig die Integration aus. Der getaktete 1-bit-Quantisierer wird mit einem Analogschalter (CD4016) und dem zweiten, als Komparator beschalteten Operationsverstärker realisiert. Aus Gründen der Übersicht wurde zur Simulation ein sinusförmiges Eingangssignal mit einer Frequenz von 1Hz und eine Abtastfrequenz von 50 Hz gewählt (Parametrierung der Impulsquelle am Analogschalter: VC=–4V ⇒ Schalter aus, VC=+4V ⇒ Schalter ein; Anstiegs- und Abfallzeit TR=TF=10ns; Impulsbreite PW=1ms; Periodendauer PER=20ms).

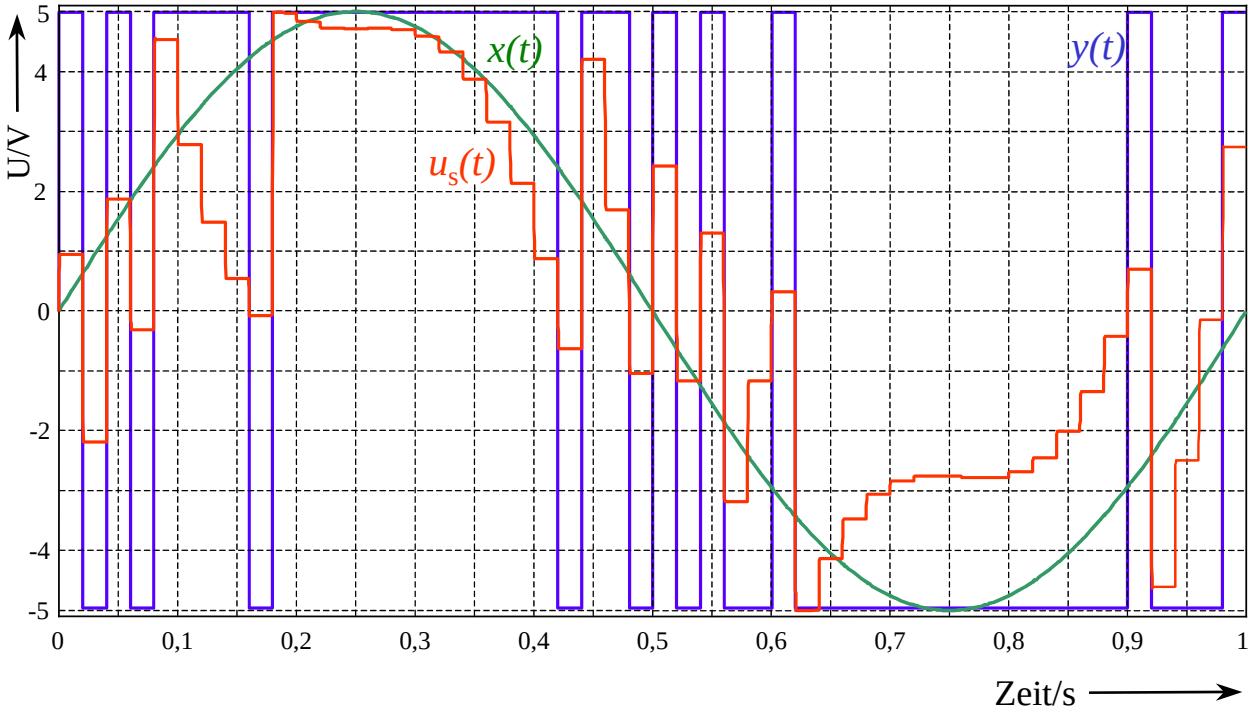


Bild 3.30: Zeitverläufe des Ein- und Ausgangsignals und des Abtastsignals $u_s(t)$

Wie aus Bild 3.30 hervorgeht, sind bei diesen Einstellungen die frequenzabhängigen parasitären Eigenschaften der Bauelemente weitgehend vernachlässigbar, so daß sich nahezu ideale Signalverläufe ergeben. Die Marker in Bild 3.29 zeigen, an welchen Stellen die in Bild 3.30 und Bild 3.31 dargestellten Zeitverläufe der verschieden Signale entnommen wurden.

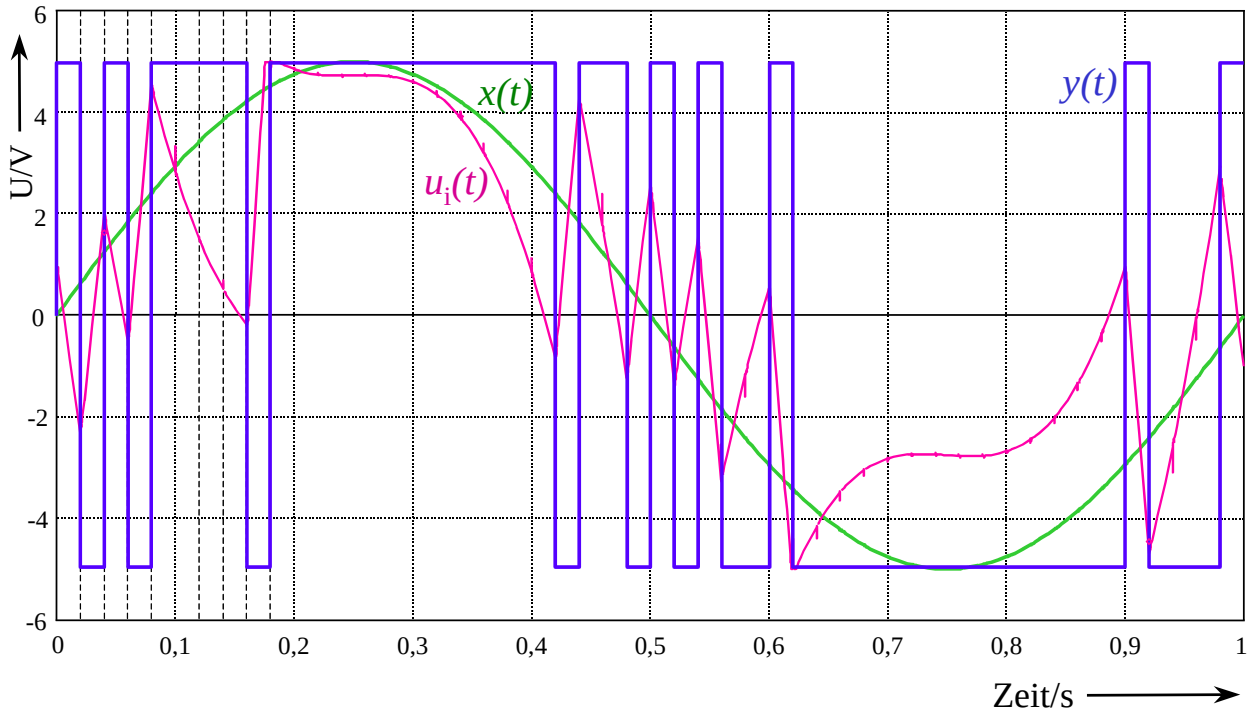


Bild 3.31: Zeitverläufe des Ein- und Ausgangsignals und des Integratorausgangssignals $u_i(t)$

Man findet die idealisierte Darstellung aus Bild 3.28 weitgehend bestätigt. Auffällig ist der recht bizarre Verlauf von $u_i(t)$ am Ausgang des Integrators. Die kleinen Nadeln, die ab und zu auf der Kurve erscheinen werden durch Übersprechen vom Schaltsignal des Analogschalters verursacht. Die genaue Analyse von $u_i(t)$ ist bei der praktischen Schaltungsrealisierung wichtig, um sicherzustellen, daß einerseits der Dynamikbereich gut ausgeschöpft wird und andererseits keine Übersteuerung vorkommt. Bei genauer Betrachtung von Bild 3.31 fällt auf, daß bei den Zeitpunkten 0,08 und ca. 0,15 kritische Stellen sind, wo schon eine Begrenzung stattfindet. Zur Korrektur wäre eine Verminderung des Verstärkungsfaktors am 1. OP nötig.

3.2.5.3 Digitale Filterung und Dezimation

Die wichtigste Folgerung aus Bild 3.26 ist, daß mit steigender Ordnung von ΣΔ-Loops die Rauschleistungsdichte bei tiefen Frequenzen verschwindend klein gemacht werden kann, wobei gleichzeitig ein kräftiger und steiler Anstieg zu hohen Frequenzen hin erfolgt. Im Zusammenspiel mit einer hohen Überabtastung kann so das Nutzband nahezu rauschfrei gemacht werden. Die zu hohen Frequenzen hin verschobene Rauschleistung muß nun durch Filterung beseitigt werden. Entscheidend ist, daß diese Aufgabe digitalen Filtern gelöst werden kann. Die Implementierung entsprechender Digitalfilter ist keineswegs sehr einfach. Zum einen stellt die wegen der Überabtastung hohe Abtastrate erhebliche Anforderungen an die Geschwindigkeit der digitalen Hardware. Audiosignale sind z.B. besonders empfindlich gegenüber Verzerrungen in Amplitude und Phase, so daß hierbei der Filteraufwand zusätzlich in die Höhe getrieben wird.

Für eine kostengünstige Realisierung sind monolithisch integrierbare Lösungen entscheidend. Dabei kann – im Hinblick auf die Zahl der benötigten äquivalenten Gatter - durchaus aufwendige Digitaltechnik zum Einsatz kommen. Zur kostensparenden und dennoch effizienten Filterung hat sich die 'Kammfiltertechnik' bewährt, weil dabei keine Multiplikationen erforderlich sind, da alle Filterkoeffizienten den Wert ***Eins*** haben.

Ein einzelnes Kammfilter ist natürlich auf keinen Fall ausreichend. Vielmehr wird die Kaskadierung zahlreicher Filterstufen benötigt.

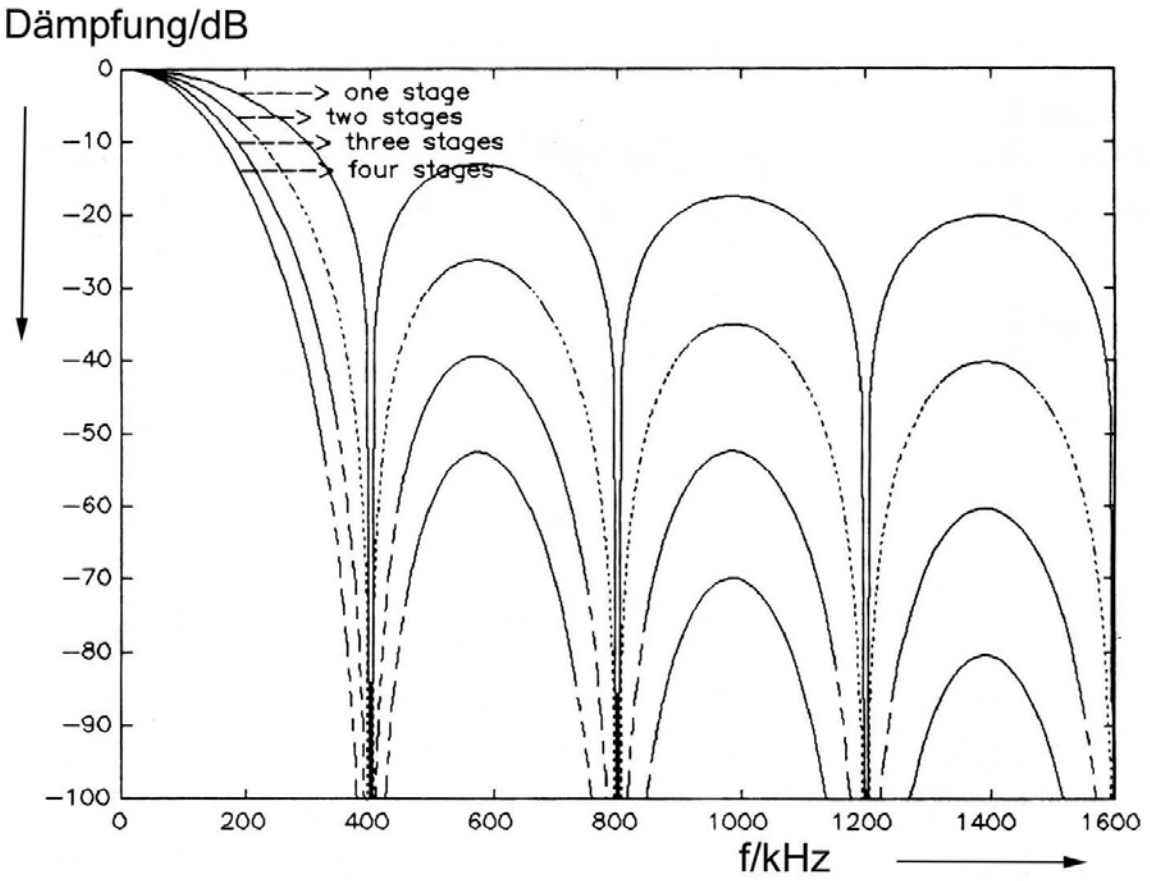


Bild 3.32: Ubertragungsfunktion bzw. Dämpfung von Kammfiltern 1. bis 4. Ordnung

3.2.5.3.1 Kammfilter und FIR–Korrekturfilter

Ein Kammfilter der Länge *N* ist ein FIR–Filter mit *N* Koeffizienten, die alle gleich **EINS** sind.

Die Übertragungsfunktion eines Kammfilters ist

$$H(z) = \sum_{n=0}^{N-1} z^{-n} = \frac{Y(z)}{X(z)}. \tag{3.52}$$

Für *N*=4 kann aus (3.52) die folgende diskrete Zeitbereichsdarstellung angegeben werden

$$y(n) = x(n) + x(n-1) + x(n-2) + x(n-3) \tag{3.53}$$

In der Tat ist ein Kammfilter ein einfacher Akkumulator, der einen „gleitenden" Mittelwert bildet. (3.52) beschreibt formal eine geometrische Reihe, so daß die Summe in der entsprechenden geschlossenen Form

$$H(z) = \frac{1 - z^{-N}}{1 - z^{-1}} = \frac{Y(z)}{X(z)} \tag{3.54}$$

dargestellt werden kann. In der Für das Beispiel N=4 kann jetzt die folgende diskrete Zeitbereichsdarstellung angegeben werden

$$y(n) - y(n-1) = x(n) - x(n-4) \quad \Rightarrow y(n) = x(n) - x(n-4) + y(n-1) \tag{3.55}$$

Das Verwenden dieser rekursiven Form führt dazu, daß die Zahl der benötigten Additionen zu Bestimmung von $y(n)$ nicht mehr von N abhängt.

Die geschlossene Form nach (3.54) legt eine Aufteilung in zwei unabhängige Prozesse nahe, nämlich eine Integration, gefolgt von einer Differentiation.

$$Y(z) = \frac{1}{1 - z^{-1}} \cdot \left[1 - z^{-N}\right] \cdot X(z) \tag{3.56}$$

Da nach dem Kammfilter eine Reduktion der Abtastrate erfolgt (N:l Dezimation) kann die Differentiation bei der niedrigeren Rate durchgeführt werden. Dies ist in Bild 3.33 angedeutet.

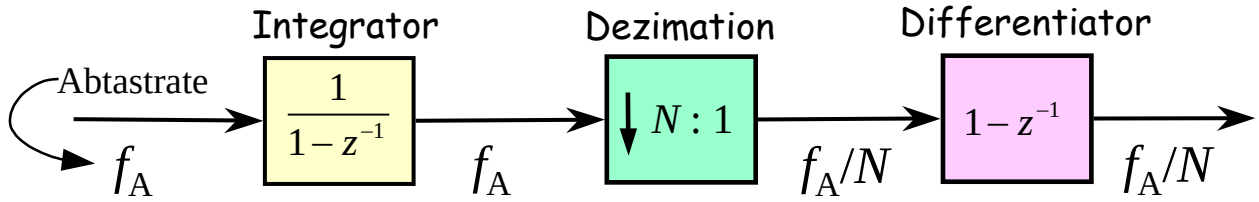


Bild 3.33: Blockdiagramm eines einstufigen Kammfilterprozesses (mit Dezimation)

In der Praxis werden stets mehrere Kammfilterstufen kaskadiert, so daß sich z.B. ein Aufbau nach Bild 3.34 ergibt. Hier sind N=16 Integrationsstufen vorhanden, auf die eine 16:1 Dezimation folgt – vgl. auch Bild 3.33. Danach wird die 16-stufige Differentiation durchgeführt. Durch das Herabsetzen der Abtastrate werden die nach Filterung verbliebenen Aliasingprodukte natürlich näher an das Nutzband herangerückt. Eine numerische Analyse zeigt, daß eine Störabstand von rund 72 dB am Ausgang des Kammfilters in Bild 3.34 vorhanden ist, was einer Auflösung von rund 12bit entspricht. An dieser Stelle kann mit der Kammfiltertechnik nicht weitergearbeitet werden, weil der „glockenförmige“ Frequenzgang im Nutzband – s. auch Bild 3.32 – zu starke Verzerrungen hervorrufen würde. Es ist deswegen nötig, den Frequenzgang durch ein nachfolgendes, komplexeres FIR-Filter – s. Bild 3.35 - zu korrigieren, das zugleich eine weitere Dezimation um den Faktor 4 durchführt. Danach hat man das Endergebnis mit 16 bit Auflösung bei einer Abtastrate von 100 kHz.

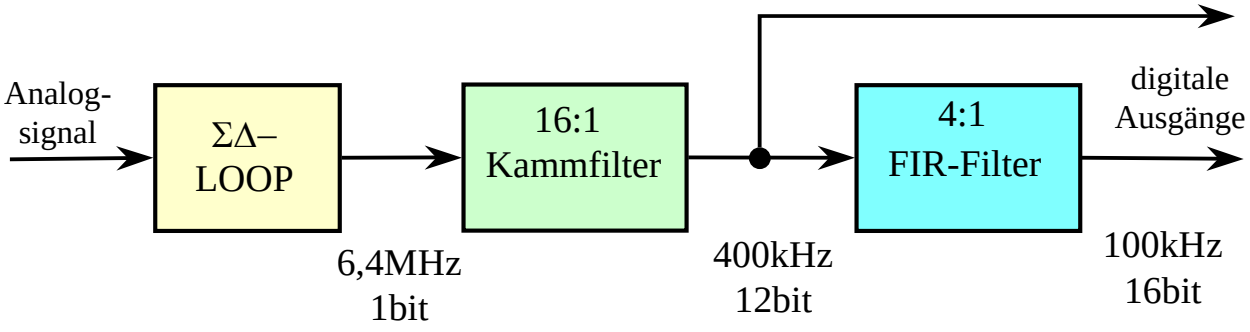


Bild 3.34: Blockschaltung der Filterung in einem typischen ΣΔ-Wandler

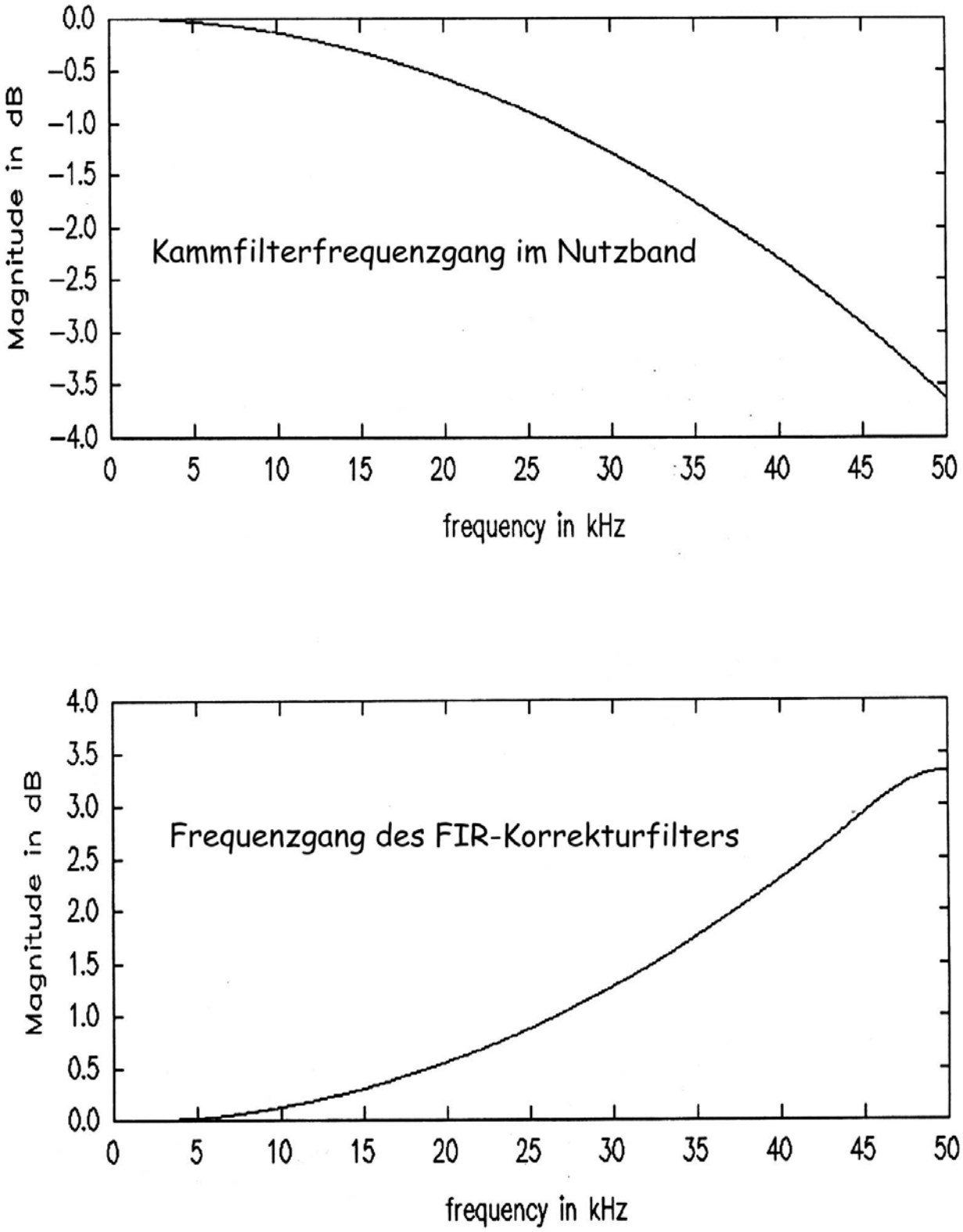
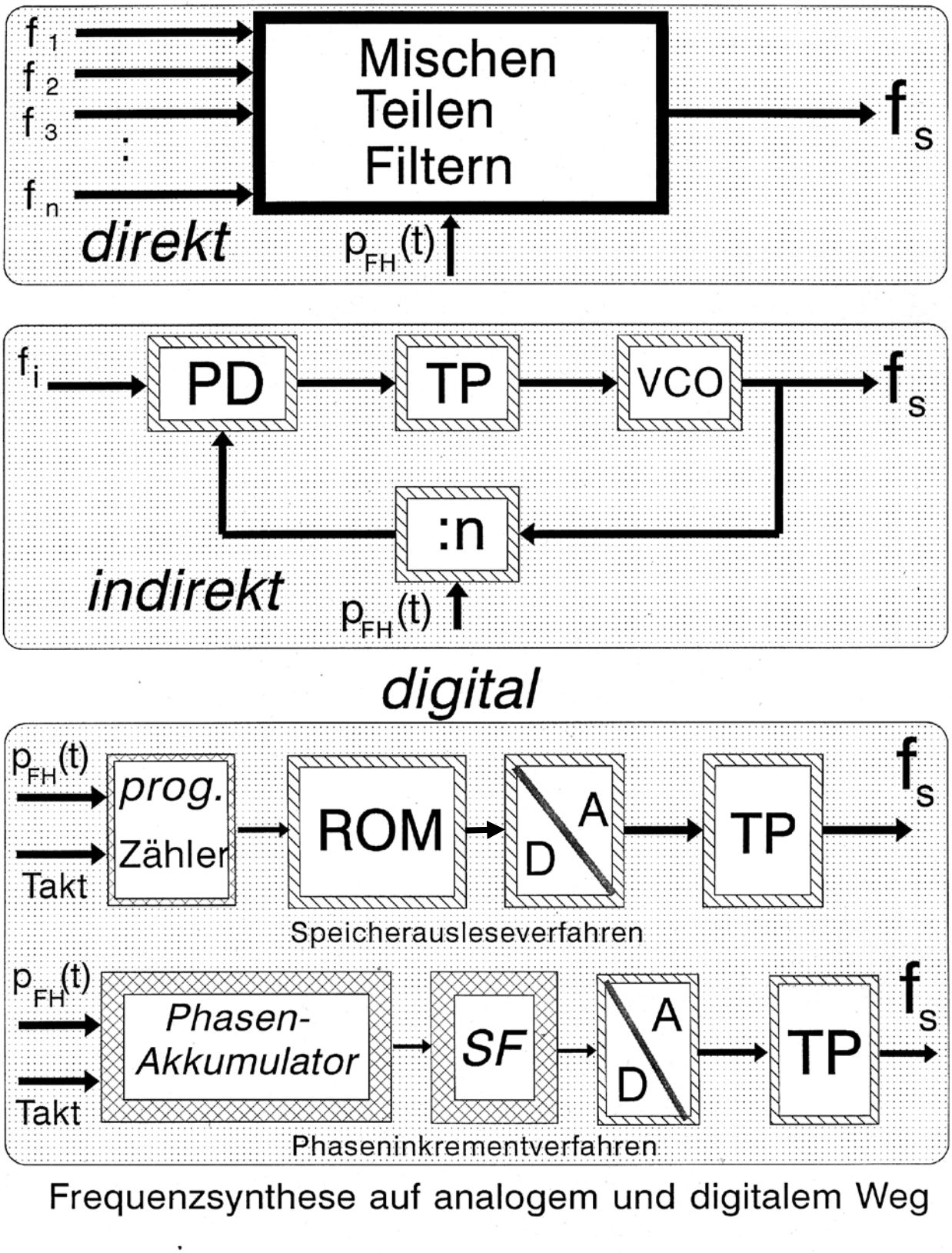


Bild 3.35: Der Kammfilterfrequenzgang im Nutzband und seine Korrektur mittels FIR-Filter

4 Digitale Frequenzsynthese

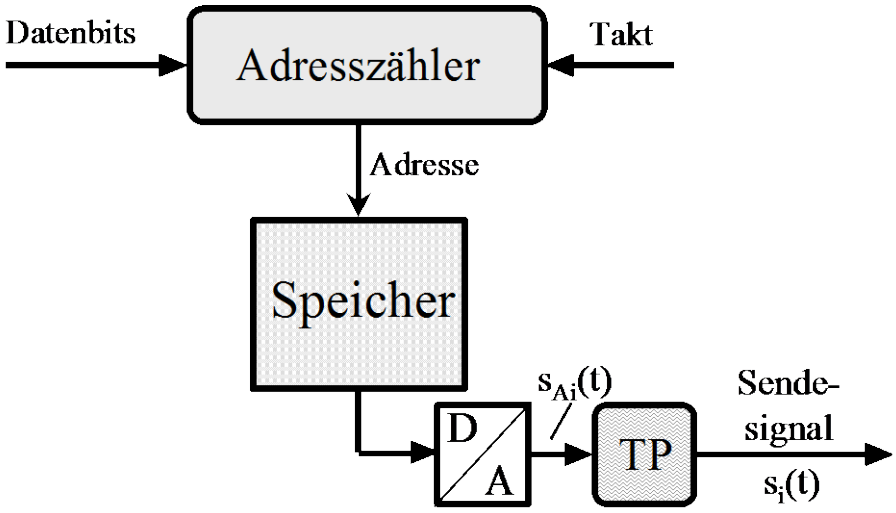


Frequenzsynthese auf analogem und digitalem Weg

Bild 4.1: Übersicht zu Verfahren der Frequenzerzeugung

4.1 Speicherausleseverfahren (Wavetable-Synthese)

Bei diesem Verfahren sind in festgelegten Adreßbereichen eines digitalen Speichers (ROM, RAM, PROM, E(E)PROM) jeweils die Abtastwerte für eine Signalform abgelegt. Die Abtastwerte gelangen auf einen D/A-Wandler. An dessen Ausgang tritt ein treppenartiges Signal $s_{Ai}(t)$ auf. Nach Tiefpaßfilterung entsteht aus $s_{Ai}(t)$ die gewünschte Signalform $s_i(t)$. Bei der Erzeugung von Signalformen für ***Frequenzsprungmodulation***[5], die hier als Beispiel dienen soll, muß der adressierte Speicherbereich im Rhythmus der Sprungrate h_r, gewechselt werden. Die einzelnen, zu Signalformen verschiedener Frequenz gehörigen Speicherbereiche, können in der Größe unterschiedlich sein.



Es gibt mehrere Lösungen zur Ansteuerung des programmierbaren Adreßzählers in Bild 4.2.

Bild 4.2: Signalsynthese durch Speicherauslesen

Wenn die Geschwindigkeitsanforderungen nicht zu hoch sind, bewährt sich der Einsatz von Mikrocontrollern aufgrund der hohen Flexibilität ihrer Timersysteme. Wenn ein Sender zu realisieren ist, kann man dem Mikrocontroller beispielsweise über eine integrierte serielle Schnittstelle unmittelbar den binären Datenstrom zuführen.

Es werden nun einige wichtige Aspekte der digitalen Frequenzsynthese quantitativ untersucht. Es geht dabei um die Optimierung des Speicherbedarfs für die Abtastwerte. Im Zusammenhang damit steht die Wahl einer günstigen Auslesetaktfrequenz, die die Abtastfrequenz f_A der zu erzeugenden Signalform darstellt. Das Verhältnis von f_A zur Frequenz f_i einer zu generierenden Signalform $s_i(t)$ beeinflußt – wie sich zeigen wird – unerwünscht die Amplitude von $s_i(t)$. Hier gilt es zu verhindern, daß sich zu hohen Frequenzen hin eine signifikante Amplitudenabnahme bemerkbar macht.

Zunächst wird die Minimierung der Zahl der zu speichernden Abtastwerte betrachtet. Wenn es darum geht, eine Signalform mit der Frequenz f_i unter Verwendung der Abtastfrequenz f_A zu synthetisieren, gilt folgende Grundgleichung:

$$\eta_i \cdot \frac{f_A}{f_i} = \left\lceil \eta_i \cdot \frac{f_A}{f_i} \right\rceil, \quad \text{mit} \quad \eta_i \in \mathbb{N}\,. \tag{4.1}$$

$\lceil \cdot \rceil$ bedeutet ganzzahliger Anteil von $(\cdot)$. Die Abtastfrequenz f_A muß zur Erfüllung des Abtasttheorems größer als $2 \cdot f_{max}$ sein, wobei f_{max} die höchste zu synthetisierende Frequenz ist.

(4.1) besagt, daß immer dann, wenn die Abtastfrequenz f_A kein ganzzahliges Vielfaches einer Frequenz f_i ist, der Bruch f_A/f_i solange mit stetig zu erhöhenden positiven ganzen Zahlen η_i zu multiplizieren ist, bis das Produkt $\eta_i \cdot (f_A/f_i)$ eine ***ganze*** positive Zahl ist. Anderenfalls würde die erzeugte Signalform unerwünschte Phasensprünge aufweisen. Der Faktor η_i ist für jede Frequenz f_i individuell festzulegen und selbstverständlich so klein wie möglich zu wählen. Wenn eine Anzahl N_{FH}

[5] engl.: Frequency Hopping ≡ ein klassisches Bandspreizverfahren mit zahlreichen „orthogonalen“ Frequenzen

orthogonaler Signalformen im Frequenzabstand der Sprungrate h_r zu generieren ist, dann sind insgesamt

$$\Pi = \sum_{i=0}^{N_{FH}-1} \eta_i \cdot \frac{f_A}{f_i} \tag{4.2}$$

Abtastwerte abzuspeichern.

Intuitiv scheint es vorteilhaft, die niedrigste Frequenz f_A als ganzzahliges Vielfaches Θ der Sprungrate h_r und die Abtastfrequenz f_A als ganzzahliges Vielfaches ς von f_0, der niedrigsten zu synthetisierenden Frequenz, zu wählen. Dann ergibt sich die Abtastwertsumme

$$\Pi = \sum_{i=0}^{N_{FH}-1} \eta_i \cdot \frac{\varsigma \cdot \Theta}{\Theta + i}, \tag{4.3}$$

für die ein Minimum zu suchen ist. Eine analytische Lösung des Problems ist nicht verfügbar. Für die Praxis wurde deshalb ein PC-Programm zur Speicherplatz-Optimierung entwickelt. Nach Eingabe der Anzahl N_{FH} der gewünschten Frequenzen, der Sprungrate h_r sollte kann - sofern die geplante Anwendung dies zuläßt - ein Variationsbereich für die niedrigste Frequenz f_0 spezifiziert werden. Dann beginnt das Programm folgende Optimierungsprozedur:

In einem ersten Durchlauf wird ς=1 gesetzt und die Summe Π wird unter Variation von f_0 in Schritten der Sprungrate h_r nach einem Minimum abgesucht. Wenn der günstigste Wert für f_0 gefunden ist, d.h. derjenige, bei dem Π möglichst klein ist, dann wird mir der folgenden Gleichung ein möglichst kleiner Wert für ς berechnet:

$$\varsigma_{\min} \geq = \frac{f_A}{f_{\max}} \cdot \left[\frac{N_{FH}-1}{\Theta} + 1 \right], \quad \text{mit} \quad \varsigma_{\min} \in \mathbb{N}. \tag{4.4}$$

(4.4) liefert nicht unbedingt für jeden Wert von $\varsigma_{\min}$ ein Minimum der Abtastwertesumme Π nach (4.3). Das beruht auf der Tatsache, daß gemeinsame Primfaktoren, die in η_i und ς enthalten sind, gekürzt werden können. Deshalb wird ein zweiter Programmdurchlauf gestartet, in dem Π wiederum auf ein Minimum hin untersucht wird, und zwar unter Variation von ς im Bereich

$$\varsigma_{\min} \leq \varsigma \leq \varsigma_{\min} + 5. \tag{4.5}$$

In der Praxis erweist es sich nicht als sinnvoll, ς über die in (4.5) angegebene Obergrenze hinaus zu vergrößern, weil der Primzahlenabstand mit größer werdenden Zahlen wächst.

Nachdem die gewünschten Frequenzen f_i feststehen und die Zahl der Abtastwerte für jede von ihnen ermittelt ist, berechnet das Programm die Abtastwerte als 8bit-Hexadezimalzahlen - gegebenenfalls auch im Zweierkomplement - und speichert sie in einem Format ab, das eine einfache EPROM-Programmierung erlaubt.

Die Abtastwerte für die Synthese der in der Tabelle 4.1 aufgelisteten 30 orthogonalen Signalformen können mit Hilfe des beschriebenen Programms mit den folgenden Vorgaben generiert werden.

- Startfrequenz f_0= 9600Hz
- Sprungrate h_r=4800Hz
- Abtastfrequenz f_A= 600kHz

Tabelle 4.1: Ein orthogonales Ensemble mit 30 Signalformen verschiedener Frequenz

1. Frequenz: 9600Hz; Sprungrate: 4800Hz; Abtastrate: 600kHz

Nr.	Frequenz in Hz	Zahl der Abtastwerte
0	9600	125
1	14400	125
2	19200	125
3	24000	25
4	28800	125
5	33600	125
6	38400	125
7	43200	125
8	48000	25
9	52800	125
10	57600	125
11	62400	125
12	67200	125
13	72000	25
14	76800	125
15	81600	125
16	86400	125
17	91200	125
18	96000	25
19	100800	125
20	105600	125
21	110400	125
22	115200	125
23	120000	5
24	124800	125
25	129600	125
26	134400	125
27	139200	125
28	144000	25
29	148800	125

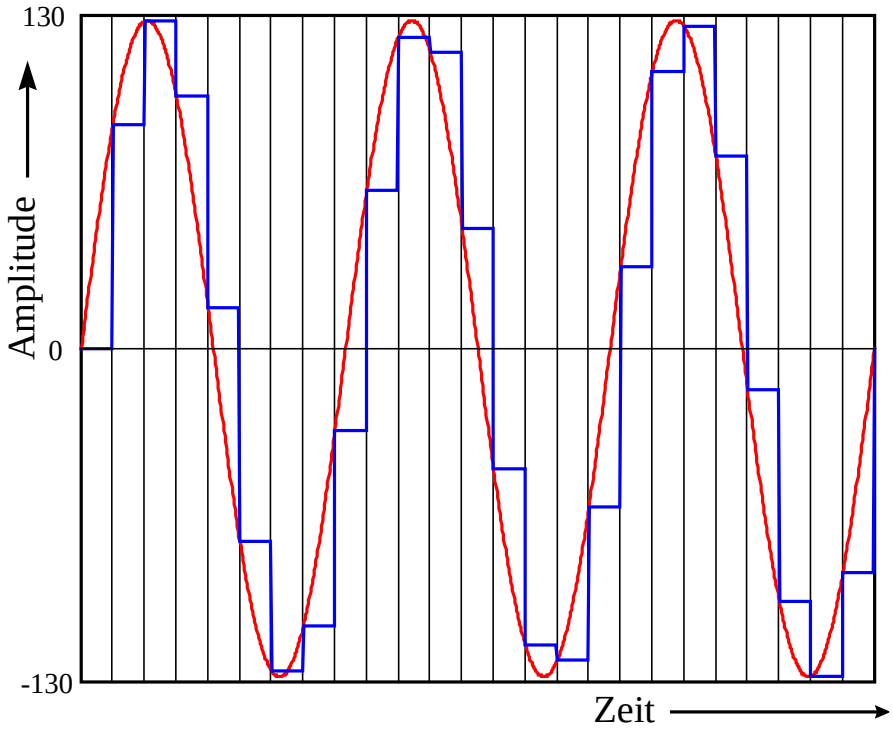
Nun wird der Einfluß des Verhältnisses f_A/f_i auf erzeugte sinusförmige Signale der Form

$$s_i(t) = \sin(2\pi f_i t). \tag{4.6}$$

untersucht. Aus Gründen der Übersichtlichkeit wurde die Amplitude in (4.6) gleich Eins gesetzt und die Zeitbegrenzung des Signals weggelassen, weil sie für die Betrachtungen hier unwichtig ist. Das Signal $s_i(t)$ ist aus abgespeicherten digitalen Abtastwerten zu generieren, wobei die Abtastwerte in einem Speicher möglichst wenig Platz beanspruchen sollen. Aus (4.2) ist zu ersehen, daß die Speichergröße proportional mit der Abtastfrequenz f_A wächst. Daher ist f_A unter Beachtung des Abtasttheorems möglichst niedrig zu halten. f_A richtet sich nach der höchsten zu erzeugenden Frequenz f_{max}, d.h. $f_A > 2 \cdot f_{max}$. Um die Anforderungen an das Rekonstruktionsfilter (Tiefpaß) niedrig zu halten ist eine Überabtastung von Vorteil.

Ein weiterer wichtiger Gesichtspunkt, der beachtet werden muß ist die Tatsache, daß die typischen Ausgangssignale eines realen D/A-Wandlers keine Diracimpulse sind, sondern aus Rechtecke der Dauer $T_A=1/f_A$.

Bild 4.3 verdeutlicht die Zusammenhänge an einem Beispiel, bei dem mit einer Abtastfrequenz f_A=600kHz eine Signalform der Frequenz f_s=72kHz erzeugt wird. Die kleinste ganze positive Zahl $\{\eta_s \cdot (f_A/f_s)\}$ ist $3 \cdot (8{,}333) = 25$. Es sind also 25 Abtastwerte zu speichern.



Das entspricht 3 Perioden eines Signals nach (4.6) mit der Frequenz f_s. In Bild 4.3 ist das typische treppenförmige Ausgangssignal, das ein realer D/A-Wandler liefert, zu sehen. Bei 8bit-Quantisierung kann es 256 Amplitudenstufen annehmen.

Zur Untersuchung des Einflusses der nichtidealen Signalausgabe in Form von Rechteck- statt Diracimpulsen wird zunächst die Abtastung eines Signals nach (4.6) mit der Abtastrate $f_A=1/T_A$ unter Einsatz eines idealen Abtasters betrachtet.

Bild 4.3: Synthese von 72kHz bei f_A=600kHz mit realem D/A-Wandler

Mit Hilfe der in (3.6) eingeführten Schreibweise für die Dirac-Impulsfolge erhält man bei Einführung des Abtastrasters T_A

$$\text{Ш}\left(\frac{t}{T_A}\right) = |T_A| \cdot \sum_{n=-\infty}^{+\infty} \delta(t - nT_A) \tag{4.7}$$

Ein ideal abgetastetes Signal $s_i(t)$ wird damit durch die zeitdiskrete und wertkontinuierliche Abtastfolge

$$s_{Ai}(t) = s_i(t) \cdot \sum_{-\infty}^{\infty} \delta(t - nT_A) = s_i(t) \cdot \frac{1}{|T_A|} \text{Ш}\left(\frac{t}{T_A}\right) \tag{4.8}$$

repräsentiert. Ein realer Abtaster kann ein solches Signal nicht liefern. Ein Abtastwert von $s_i(t)$, der von einem realen Abtaster zu einem Zeitpunkt $n \cdot T_A$ genommen wird, bleibt bis zum Zeitpunkt $(n+1) \cdot T_A$ konstant. Ein real abgetastetes Signal entspricht somit der in Bild 4.3 dargestellten Treppenfunktion, und wird durch

$$s_{re}(t) = s_i(t) \cdot \frac{1}{|T_A|} \cdot \text{Ш}\left(\frac{t}{T_A}\right) * \text{rect}\left(\frac{t - \frac{T_A}{2}}{T_A}\right) \tag{4.9}$$

beschrieben, d.h. um jeden Diracimpuls ist ein Rechteck der Breite T_A gruppiert.

Die weiteren Untersuchungen werden im Frequenzbereich durchgeführt. Dazu wird die Fouriertransformierte von (4.9) bestimmt. Mit $s_i(t) \circ\!\!-\!\!\bullet\, S_i(f)$ und (3.14) erhält man

$$s_{re}(t) \quad \circ\!\!-\!\!\bullet \quad S_{re}(f) = \left[S_i(f) * \text{Ш}(f \cdot T_A)\right] \cdot T_A \cdot \text{si}(\pi f T_A) \cdot \text{e}^{-j\pi T_A} \tag{4.10}$$

Nach (3.14) ist $\text{Ш}(f T_A) = \frac{1}{|T_A|} \cdot \sum_{n=-\infty}^{+\infty} \delta\left[f - \frac{n}{T_A}\right]$, so daß

$$S_{re}(f) = \left[S_i(f) * \sum_{n=-\infty}^{+\infty} \delta\left[f - \frac{n}{T_A}\right]\right] \cdot \text{si}(\pi f T_A) \cdot \text{e}^{-j\pi f T_A} \tag{4.11}$$

folgt. Aufgrund der „Siebeigenschaft" des Diracimpulses erscheint $S_i(f)$ nur an den diskreten Auftrittsstellen der Diracimpulse, d.h. man hat

$$S_{re}(f) = \left[\sum_{n=-\infty}^{+\infty} S_i\left[f - \frac{n}{T_A}\right]\right] \cdot \text{si}(\pi f T_A) \cdot \text{e}^{-j\pi f T_A}. \tag{4.12}$$

Wendet man zur Bestimmung von $S_i(f)$ auf (4.6) die Fouriertransformation an, so ergibt sich

$$s_i(t) \circ\!\!-\!\!\bullet\, S_i(f) = \frac{1}{2}\left[\delta(f + f_i) - \delta(f - f_i)\right] \cdot \text{e}^{j\pi/2} \tag{4.13}$$

Damit wird aus (4.12)

$$S_{re}(f) = \frac{1}{2}\left[\sum_{n=-\infty}^{+\infty} \delta\left(f + f_i - \frac{n}{T_A}\right) - \delta\left(f - f_i - \frac{n}{T_A}\right)\right] \cdot \text{e}^{j\pi/2} \cdot \text{si}(\pi f T_A) \cdot \text{e}^{-j\pi f T_A}. \tag{4.14}$$

In (4.14) erkennt man, daß aufgrund der nichtidealen Abtastung eine frequenzabhängige Gewichtung des Spektrums durch die Funktion $\text{si}(\pi f \cdot T_A)$ eintritt. Dieser Effekt muß soweit wie möglich eliminiert werden, weil man bei der Signalformsynthese Wert auf gleiche Amplituden bei allen Frequenzen legt. Die Exponentialfunktionen $\text{e}^{j\pi/2}$ und $\text{e}^{-j\pi f T_A}$ in (4.14) stellen Phasenverschiebungen dar und können, ebenso wie der Faktor 1/2, bei den weiteren Betrachtungen ohne Beschränkung der Allgemeinheit außer acht gelassen werden.

Zur Verdeutlichung der Zusammenhänge wird ein Beispiel mit der Abtastfrequenz $f_A = 3f_i$ untersucht.

Bild 4.4 dient zur Veranschaulichung des Einflusses der Amplitudengewichtung auf das Ausgangssignal eines Frequenzsynthesizers nach Bild 4.2. Die senkrechten Pfeile zeigen die Lage derjenigen Diracimpulse aus (4.14), die innerhalb der Bandbreite eines idealen Tiefpasses mit der Grenzfrequenz f_g liegen, wobei $f_i=1/3f_A< f_g$ ist. Man sieht, daß eine Gewichtung der Amplituden in Form der Funktion si($\pi f \cdot T_A$) vorliegt. Der Tiefpaß (schattiertes Rechteck) unterdrückt die - in Bild 4.4 nicht sichtbaren - Spektrallinien von $S_{re}(f)$. Durch die Funktion si($\pi f \cdot T_A$) wird die gewünschte Spektrallinie bei f_i in der Amplitude beeinflußt. Im Beispiel mit $f_A=3 \cdot f_i$ ist der Gewichtungsfaktor si($\pi/3$)=0,826 und damit nicht mehr vernachlässigbar. Im folgenden wird deshalb eine einfache digitale Korrekturmöglichkeit angegeben.

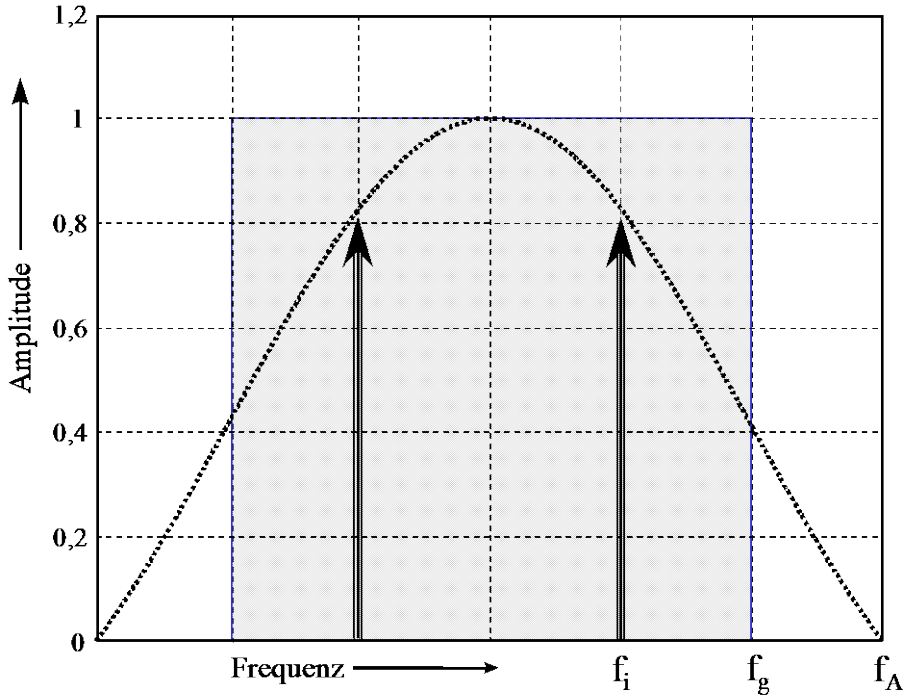


Bild 4.4: Synthesebeispiel für $f_A=3f_i$

Wenn - ausgehend von dem Beispiel nach Bild 4.3 - f_{max}=140 kHz die höchste zu erzeugende Frequenz wäre, dann würde über einen Frequenzbereich von 30 kHz bis 140 kHz die Amplitude bis auf si($\pi \cdot 140/600$)≈0,912, d.h. um etwa 8,8% absinken. Aufgrund der direkten Abspeicherung der Abtastwerte zur digitalen Synthese ist es leicht möglich, an dieser Stelle eine Korrektur durchzuführen. Für jede zu synthetisierende Frequenz f_i kann ein Korrekturfaktor für die zugehörigen Abtastwerte in der Form

$$K_{fi} = \frac{\mathrm{si}\left(\pi f_{\max} T_A\right)}{\mathrm{si}\left(\pi f_i T_A\right)} \tag{4.15}$$

berechnet werden, mit dem jeder Abtastwert der jeweiligen Frequenz zu gewichten ist. Bei der tiefsten Frequenz (30 kHz) sind demnach alle Abtastwerte mit K_{30}≈0,912 zu multiplizieren, also um 8,8% zu erniedrigen. Zu höheren Frequenzen hin vergrößert sich der Faktor K_{fi}. So ist z.B. für 100 kHz $K_{100} \approx 0{,}956$, d.h. die Abtastwerte dieser Frequenz sind nur noch um rund 4,4% zu erniedrigen. Bei 140kHz wird schließlich K_{140}=1.

Das hier untersuchte Syntheseverfahren kommt an seine Grenzen, wenn es entweder darum geht, bestimmte „krumme“ Frequenzwerte sehr genau zu treffen, d.h. wenn etwa die beiden letzten Dezimalstellen in Tabelle 4.1 nicht mehr Null sind, sondern beliebige Werte annehmen sollen, oder wenn ein Durchlaufen bestimmter Frequenzbänder in feinen Stufen erforderlich ist. Letzteres ist z.B. beim elektronischen Abstimmen von Rundfunk- oder Fernsehempfängern der Fall, wobei der lokale Mischoszillator seine Frequenz in hinreichend kleinen Schritten variieren muß. Beim UKW-Rundfunk mit einer Kanalbandbreite von 200kHz beträgt die typische Schrittweite 50kHz, wobei die Mischfrequenz f_M>100MHz ist.

Ein einfaches Beispiel unter Verwendung von (4.1) macht sofort klar, wie schnell das Speicherausleseverfahren bei „krummen“ Frequenzwerten versagt. Will man mit einer Abtastfrequenz f_A=600kHz eine Signalform der Frequenz f_s=133kHz erzeugen, dann liefert die Division f_A/f_s den Zahlenwert 4,511278195488721804. Hierfür läßt sich offenbar keine Periodizität mehr finden, so daß Abtastwerte für die gesamte Dauer der Signalform im Speicher abzulegen wären.

Wenn es also darum geht, beliebige Frequenzwerte in sehr kleinen Schritten zu erzeugen, bzw. auch relativ große Frequenzbereiche mit feiner Auflösung zu durchfahren, kommt eine ganz andere Methode der Frequenzsynthese zur Anwendung, die unter der Bezeichnung „Phasenakkumulation“ bekannt ist.

4.2 Frequenzsynthese durch Phasenakkumulation

Beim Begriff Sinusschwingung denkt man normalerweise an eine Gleichung der Form $s(t) = A \cdot \sin(2 \cdot \pi f \cdot t)$. Man hat es bei dieser technisch äußerst wichtigen Signalklasse leider mit nichtlinearen Funktionen zu tun, die sich auf den ersten Blick nicht einfach auf digitalem Weg erzeugen lassen. Man könnte zunächst an eine Konstruktion aus stückweise linearen Elementen denken, die man schließlich soweit verkleinern kann, daß quasi-kontinuierliche Verläufe entstehen. Die Winkelbeschreibung im Argument der Sinusfunktion ist – wie Bild 4.5 in einfacher Weise verdeutlicht - ein solches „lineares Element“. Denn pro Zeitintervall wird stets ein gleich großer Winkelbereich durchlaufen, ein Sachverhalt, der durch die Winkelgeschwindigkeit $\omega=2\pi f$ beschrieben wird.

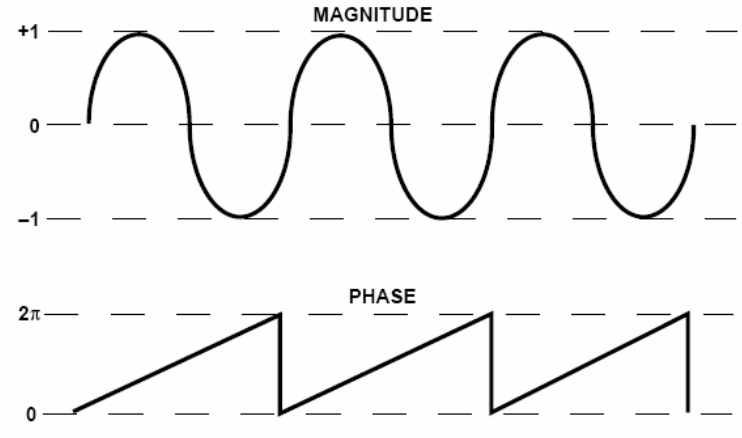


Bild 4.5: Phase und Amplitude einer Sinusschwingung

Ausgehend von dieser Überlegung kann eine Phasendrehung $\Delta\varphi$ pro Zeitintervall definiert werden:

$$\Delta\varphi = \omega dt \quad \Rightarrow \quad \omega = \frac{\Delta\varphi}{dt} = 2\pi f \tag{4.16}$$

Für die angestrebte digitale Realisierung muß das Zeitintervall *dt* diskret sein, was sich leicht mit Hilfe eines hochfrequenten Taktes f_c bewerkstelligen läßt. Setzt man $dt=1/f_c$, dann erhält man aus (4.16) die Frequenz

$$f = \frac{\Delta\varphi \cdot f_c}{2\pi} \tag{4.17}$$

Periodische kontinuierliche Signale weisen immer einen Phasenbereich von 0…2π auf. Bei Überschreitung dieses Bereichs wiederholt sich der Verlauf, d.h. man hat ein „Modulo-2π“-Verhalten. Eine digitale Implementierung muß sich prinzipiell genauso verhalten. Das erreicht man dadurch, daß der Bereich 0...2π auf eine große Binärzahl mit 32...48 bit abgebildet wird. Eine derart hohe Stellenzahl ist – wie im weiteren noch deutlich wird – für eine hohe Frequenzauflösung bei der Synthese erforderlich. Für marktübliche Bausteine z.B. AD7008 (Analog Devices) sind mindestens 32 bit die Regel. Damit wird – anschaulich gesprochen – der „Einheits-Phasenkreis“ mit dem Umfang 2π in 2^{32} digitale Schritte unterteilt – s. auch Bild 4.6 – so daß (4.17) entsprechend umgeschrieben werden kann:

$$f = \frac{\Delta\varphi \cdot f_c}{2^{32}}. \tag{4.18}$$

Nachdem der lineare Teil für die Signalsynthese bereitsteht, muß noch die nichtlineare Zuordnung der Amplitude zum Phasenwert vorgenommen werden. Das geht in digitaler Hardware am einfachsten über eine Wertetabelle, d.h. der Inhalt des Phasenakkumulators stellt eine Adresse dar, unter der in einem digitalen Speicher der zugehörige Amplitudenwert abgelegt ist. Die praktische Realisierung wird weiter unten näher betrachtet.

Damit man hohe Frequenzen synthetisieren kann muß natürlich f_c auch entsprechend hoch gewählt werden. Genauer gesagt ist aufgrund des Abtasttheorems die höchste Frequenz $f_{max}<f_c/2$. In der Praxis liegen übliche Werte von f_c im Bereich 50 MHz ...300 MHz, d.h. man kann damit Synthese mit Obergrenzen von ca. 25 MHz...150 MHz betreiben. Neben der höchsten synthetisierbaren Frequenz ist natürlich auch die niedrigste und damit die erzielbare Schrittweite von Interesse.

Mit f_c=50 MHz und einer Phasenakkumulatorlänge von 32 bit erhält man in einfacher Weise aus (4.18) für $\Delta\varphi$=1, d.h. die kleinstmögliche Schrittweite

$$f_{min} = \frac{50.000.000}{2^{32}}\,\text{Hz} = 0{,}01164153218\ \text{Hz} \tag{4.19}$$

Die Frequenzauflösung beträgt also ca. 11,6mHz, so daß die numerischen Fehler immer gegenüber den Toleranzen der Quarze, mit denen man f_c erzeugt, vernachlässigbar sind.

Wenn man z.B. f_s=1 MHz erzeugen will, errechnet sich mit obigen Daten die Phasenschrittweite zu

$$\Delta\varphi = \frac{2^{32}}{50 \cdot 10^6} \cdot 10^6 = 2^{32}/50 = 85899345{,}92 \equiv 5\ 1E\ B8\ 52H \tag{4.20}$$

Da nur ganze Zahlen verarbeitet werden, entsteht ein Rundungsfehler, der sich so auswirkt, daß die Frequenz f_{sa}=1.000.000,000931 synthetisiert wird. Der Fehler ist also kleiner als 1mHz!

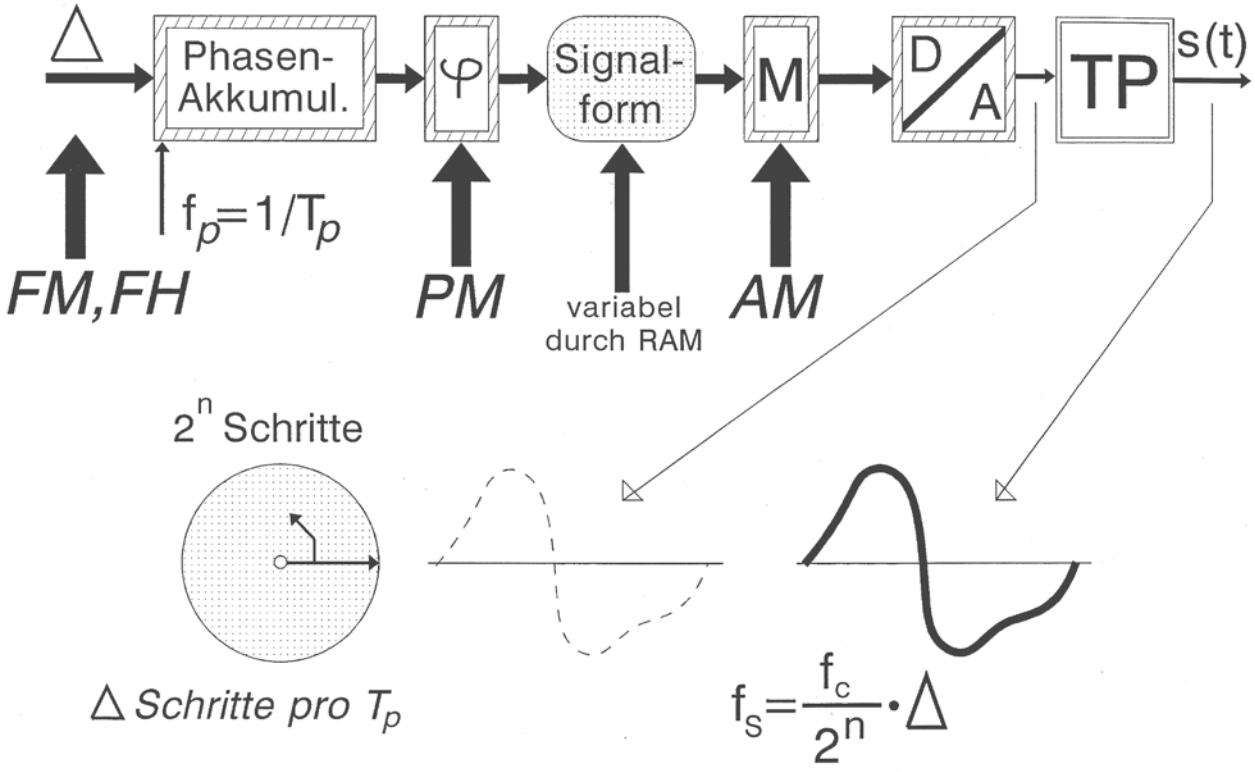


Bild 4.6: Vom Phasenakkumulator zum ***numerisch kontrollierten Oszillator*** (NCO)

Wie Bild 4.6 veranschaulicht, kann auf der Grundlage des Phasenakkumulationsprinzips nicht nur reine Frequenzsynthese betrieben werden, sondern man kann auf relativ einfache Weise auch komplexe digitale Modulationsverfahren realisieren, die z.B. durch die Gleichung

$$s(t) = A(t) \cdot \mathbb{F}\left[2\pi f(t) + \varphi(t)\right] \text{mit } \mathbb{F}(\bullet) \text{ über Wertetabelle} \qquad (4.21)$$

beschrieben werden. (4.21) schließt Amplituden-, Frequenz- und Phasenmodulation ein. Zudem könnte eine beliebige Ausgangssignalform über eine Wertetabelle generiert werden, die bei Verwendung eines RAM auch noch dynamisch veränderbar wäre.

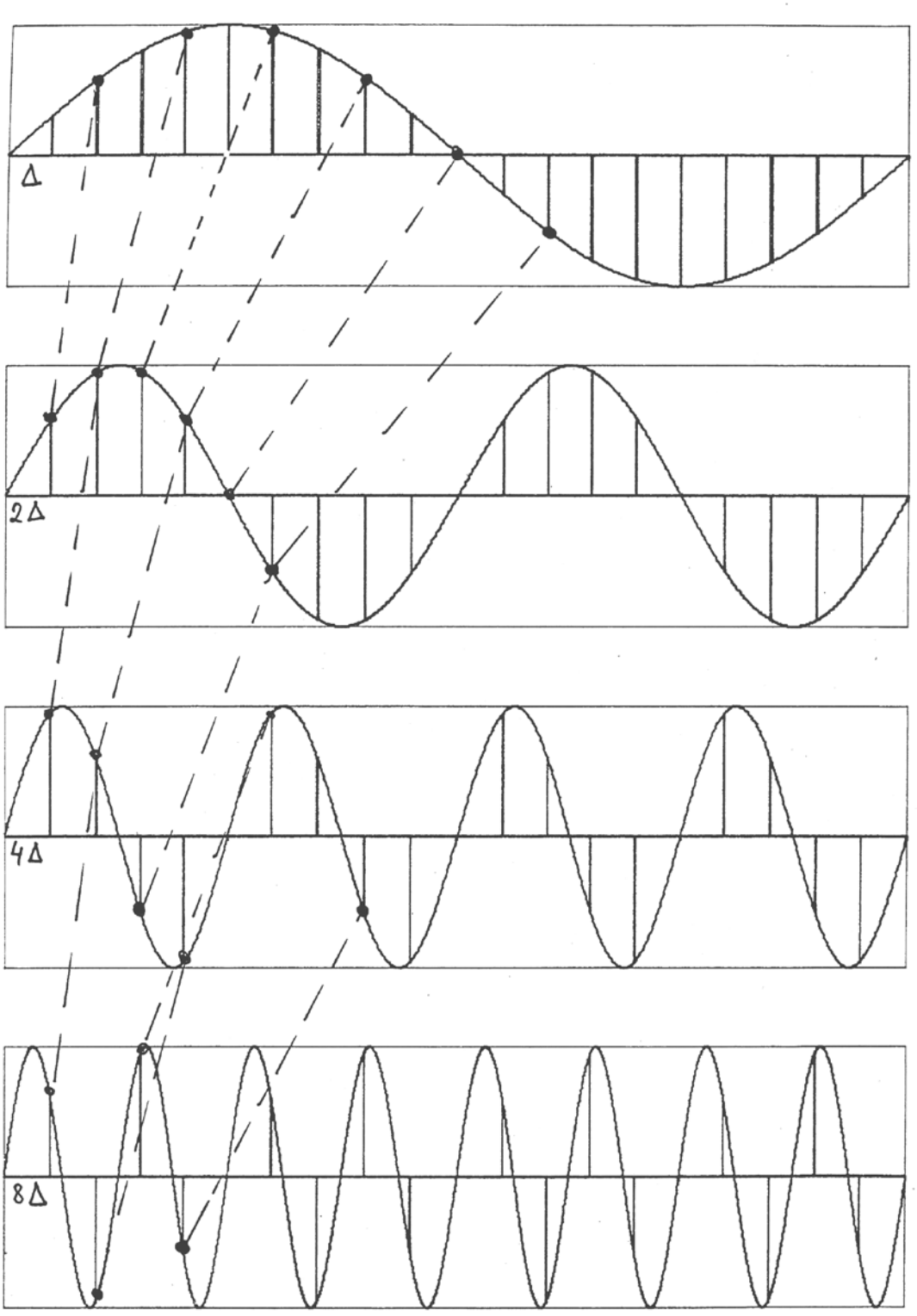


Bild 4.7: Illustration der Arbeitsweise des Phasenakkumulators bei der Frequenzsynthese

Bevor die Verwendung des NCO-Prinzips bei Modulationsverfahren weiter vertieft wird, ist noch ein Blick auf die Signalformerzeugung wichtig. Anhand von Bild 4.7 läßt sich anschaulich erklären, wie durch Verdopplung der Phasenschrittweite jeweils eine Verdoppelung der Ausgangsfrequenz erfolgt. Aufgrund der großen Akkumulatorlänge (32...48bit) ist aber auch eine äußerst feine Stufung möglich, wie die Rechenbeispiele (4.19) und (4.20) gezeigt haben. Ein oberes Ende ist erreicht, wenn $\Delta\varphi$ die halbe Akkumulatorlänge hat. In diesem Fall gibt es nur noch zwei Abtastwerte pro Periode, so daß das Abtasttheorem gerade noch erfüllt wäre. Eine einfache Überlegung anhand der untersten Grafik von Bild 4.7 macht deutlich, daß dies kein technisch brauchbarer Betrieb mehr wäre, denn die beiden Abtastwerte könnten exakt auf zwei Nulldurchgänge fallen.

Eine weitere offene Frage ist noch die Organisation des Signalformspeichers. Aufgrund der enormen Wortbreite des Phasenakkumulators ist es technisch nicht sinnvoll, diesen in voller Länge auf eine Adresse abzubilden, weil dann z.B. für 32 bit immerhin 4.294.967.296 Abtastwerte pro Sinusperiode anfallen würden – wohingegen schon 2 ausreichen würden, um das Abtasttheorem zu erfüllen. Diese einfache Betrachtung macht klar, daß viele Abtastwerte ausgelassen werden können, ohne daß dadurch ein Fehler entsteht. Zur Verdeutlichung sei noch mal der Blick auf die obere Grafik von Bild 4.7 gelenkt. Hier sind 21 Abtastwerte pro Periode vorhanden, so daß sich deren Zahl ohne weiteres durch 7 teilen, d.h. auf 3 reduzieren ließe. In der Praxis wird diese „Dezimation“ beim Phasenakkumulator einfach dadurch bewerkstelligt, daß man eine gewisse Anzahl niederwertiger Bits unberücksichtigt läßt, d.h. nur den oberen Teil des „Phasenvektors“ in einer Länge von ca. 10..12 bit zur Adreßerzeugung benutzt. Es ist wichtig, sich klarzumachen, daß dadurch die Frequenzauflösung nicht beeinträchtigt wird, da diese einzig und allein von der Schrittweite beim Durchlaufen des Phasenkreises abhängt und nicht von der „Dichte“ bzw. der Zahl der Abtastwerte pro Periode. Der zweite Gesichtspunkt bei der Organisation des Signalformspeichers ist die Auflösung der ausgegebenen Worte, die der D/A-Wandlung zugeführt werden. Es macht rein numerisch natürlich keinen Sinn, die Auflösung höher zur treiben als die maximale Zahl der Abtastwerte pro Periode. D.h. zu einer 10 bit-Adresse paßt eine Auflösung von 10 bit oder zu einer 12 bit-Adresse passen 12 bit Auflösung.

Es gibt eine Reihe von Halbleiterherstellern, die numerisch kontrollierte Oszillatoren (NCO) mit zusätzlicher Peripherie zur Erzeugung komplexer Modulationssignale entsprechend (4.21) auf rein digitalem Wege versehen haben und solche hochintegrierten Schaltungen unter der Bezeichnung DDS (≡ Direct Digital Synthesizer) anbieten. Der schon erwähnte AD7008 ist einer der ältesten Vertreter dieser Gattung und bereits seit über 15 Jahren auf dem Markt.

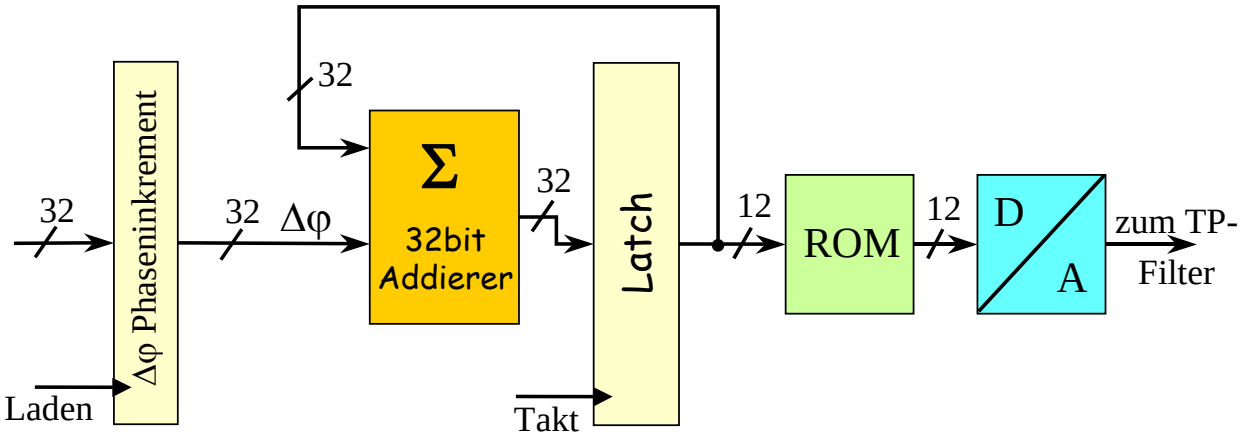


Bild 4.8: Detaillierter Aufbau des Phasenakkumulators bis zur Analogsignalerzeugung

Bevor dieser Baustein anhand von Bild 4.9 näher beschrieben wird, sind zunächst einige Details des Kernstücks, nämlich des Phasenakkumulators zu erläutern. Bild 4.8 zeigt die prinzipielle

Struktur des Kernstücks eines DDS. Am Eingang befindet sich ein Register der Breite 32 bit, in dem die Phasenschrittweite Δφ abgelegt wird. Das Einschreiben muß mit dem Haupttakt synchronisiert werden, damit es nicht zu ungewollten Phasensprüngen im dynamischen Betrieb kommt. Im Datenblatt des AD7008 ist die recht aufwendige Synchronisationseinrichtung ausführlich beschrieben. Wie man aus Bild 4.8 ersieht, gelangt das Phaseninkrement Δφ auf einen 32 bit Addierer (genauer gesagt ist dies ein Modulo-32bit-Addierer, der keine Übertrage bildet sondern überläuft, d.h. beim Rest $Z\text{-}2^{32}$ weitermacht.) Der zweite Eingang des Addierers erhält den Latchinhalt und mit jedem Takt wird eine neue Summe ins Latch geschrieben. Mit dem Laden des Phaseninkrements Δφ ist der DDS somit für die permanente Ausgabe einer Frequenz nach (4.18) programmiert. Wie oben erläutert, ist es bei Verwendung eines 12 bit D/A-Wandlers sinnvoll, den Signalformspeicher (ROM) sowohl in der Adreßwortbreite als auch in der Datenwortbreite auf 12 bit auszulegen. Vom Ausgang des Latches werden daher nur noch die 12 MSBs als Adresse an das ROM weitergegeben. Da bei niedrigen Frequenzen, d.h. für $f \ll f_A/2$, eine hohe Überabtastung vorliegt, sind die Anforderungen an den Rekonstruktionstiefpaß entsprechend gering. Nähert sich allerdings die zu synthetisierende Frequenz der Nyquistfrequenz, dann wird eine steilflankige und damit aufwendige Filterung erforderlich – näher Betrachtungen dazu folgen noch.

Wie man aus Bild 4.9 ersieht, verfügt der AD7008 über zwei eingangsseitige Speicher für Phaseninkremente Δφ. Dadurch ist über das Steuersignal „FSELECT“ ein schneller Frequenzwechsel möglich, was z.B. für die Realisierung eines FSK-Senders mit hoher Datenrate von Vorteil wäre.

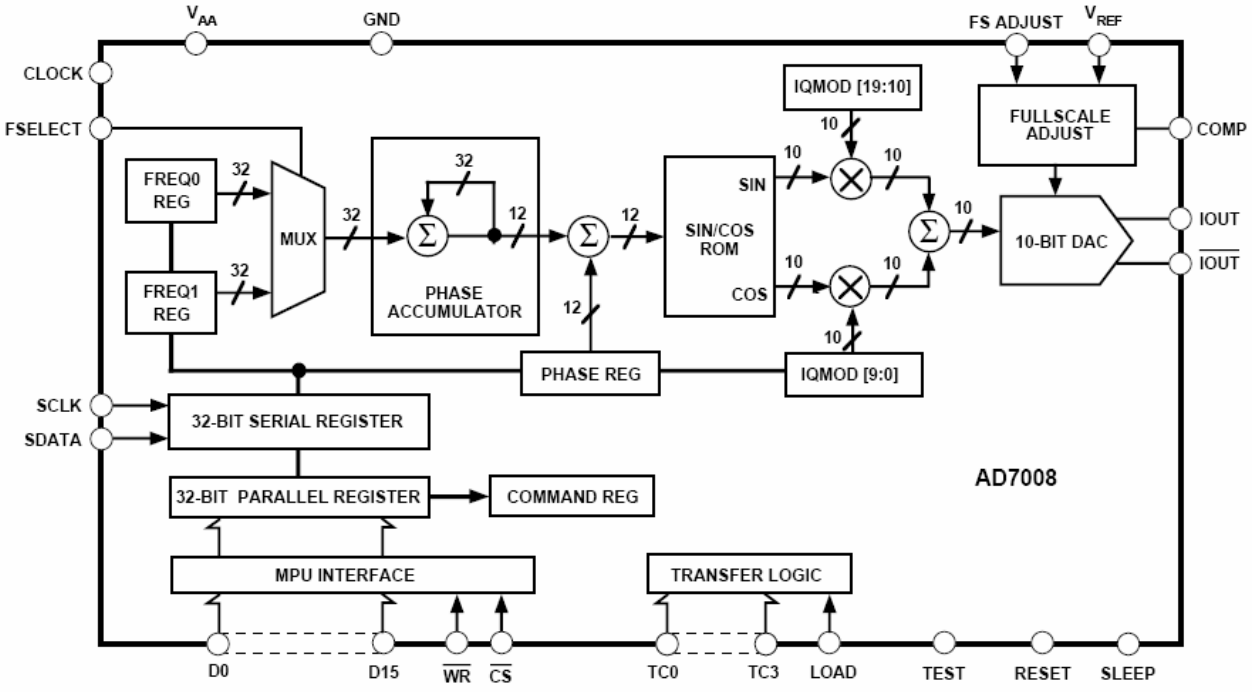


Bild 4.9: Funktionsblockschaltbild des DDS vom Typ AD7008 (Analog Devices)

Neben schneller Frequenzsprungmodulation erlaubt der AD7008 auch die direkte Phasenmodulation. Dies geschieht in einfacher Weise über einen Modulo-4096-Addierer (12 bit), der zwischen Latch und Signalformspeicher geschaltet ist. In ein 12 bit–Register „PHASE REG“ wird dazu der gewünschte „Phasenhub“ eingetragen. Da hier die Wortbreite bereits auf 12 bit begrenzt ist, bedeutet z.B. das Eintragen von 2048 im „PHASE REG“ eine Phasenverschiebung von 180°; bei den Zahlenwerten 1024 oder 3072 hätte man +90°, bzw. –90°.

Ein weiterer Schritt zur Realisierung der universellen Modulatorfunktionen gemäß (4.21) folgt am Ausgang des Signalformspeichers in Form digitaler Multiplizierer zur Amplitudenbeeinflussung.

Es handelt sich bei dem in Bild 4.9 dargestellten Aufbau um eine so genannte Quadraturmodulatorstruktur[6], bei der die ausgegebene Signalform gleichzeitig in Sinus- und in Cosinuslage vorhanden ist, und wobei die Amplituden der beiden Anteile individuell über zwei Hälften des Registers „IQMOD“ eingestellt werden können. Da die SIN- und COS–Abtastwerte jeweils als 10bit Zweierkomplementzahlen vorliegen, ist „IQMOD“ entsprechend in zwei 10bit-Hälften ([0..9] für COS und [10...19] für SIN) aufgeteilt. Die vorzeichenbehafteten Produkte werden jeweils auf 10bit begrenzt und addiert. Das ebenfalls auf 10 bit begrenzte Ergebnis gelangt schließlich auf den D/A-Wandler. Da keine Überläufe verarbeitet werden, muß beim Programmieren auf entsprechende Skalierung geachtet werden, damit keine Sprünge im Signalverlauf auftreten.

Am Pin “FSADJUST” (Full-Scale Adjust Control) wird ein Widerstand R_{adj} angeschlossen, der zur analogen Masse AGND führt. Dadurch wird der maximale Ausgangsstrom des D/A-Wandlers festgelegt. Dabei gilt die Beziehung

$$I_{DAC}/\mathrm{mA} = \frac{6233 \cdot U_{ref}/\mathrm{V}}{R_{adj}/\Omega} \tag{4.22}$$

Mit U_{ref}=1,27V und R_{adj}=390Ω erhält man I_{DAC}≈20mA, so daß sich bei Anschluß eines Widerstandes von 50Ω an den Ausgängen IOUT bzw. $\overline{\mathrm{IOUT}}$ jeweils eine Spannung von $1V_{ss}$ einstellt.

Die Eingabe der Programmierdaten für den DDS kann seriell über die Pins „SDATA“ und „SCLK“ oder parallel über einen 16 bit breiten Datenbus D0...D15 erfolgen. Zwecks Kompatibilität mit 8-bit-Mikrocontrollern kann über das „COMMAND REG“ auch ein 8 bit Datentransfer eingestellt werden, bei dem die oberen 8 bit (D8...D15) des Datenbusses irrelevant sind.

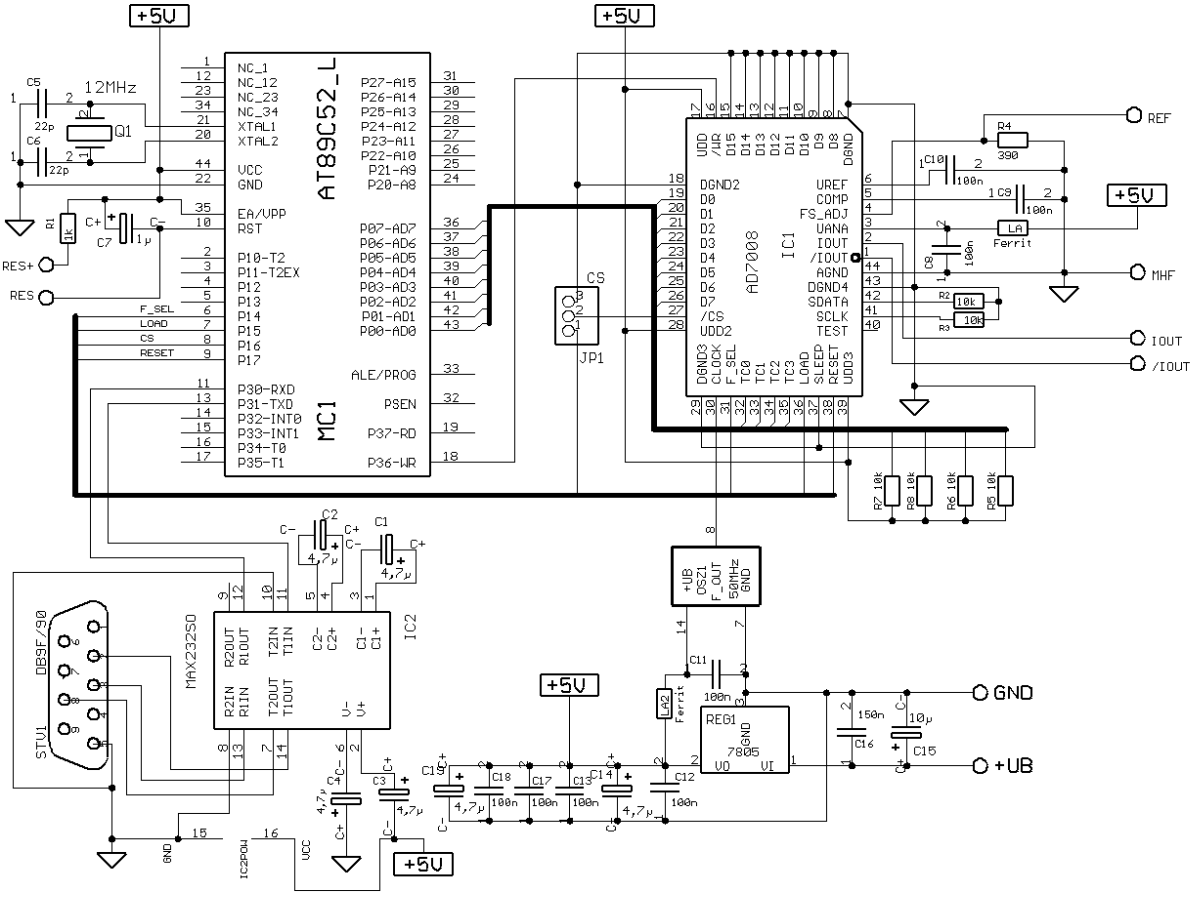


Bild 4.10: Verbindung des DDS AD7008 mit einem 8 bit–Mikrocontroller

[6] Betrachtungen zu Sinn und Zweck der Quadraturmodulation folgen weiter unten

Wie die Schaltung in Bild 4.10 zeigt, kann ein DSS sehr einfach mit einem Mikrocontroller vom Typ 8052 verbunden werden. Im Mikrocontroller können neben einfachen Routinen zur Erzeugung verschiedener Frequenzen auch Programme für komplexe Vektormodulation abgelegt werden.

Die serielle Schnittstelle des MC ermöglicht darüber hinaus eine flexible und übersichtliche Programmierung von einem PC oder Notebook aus.

Ein einfacherer, jedoch etwas schnellerer DDS Baustein, der bereits in zahlreichen Forschungsprojekten am IIIT mit Erfolg eingesetzt wurde ist der AD9850.

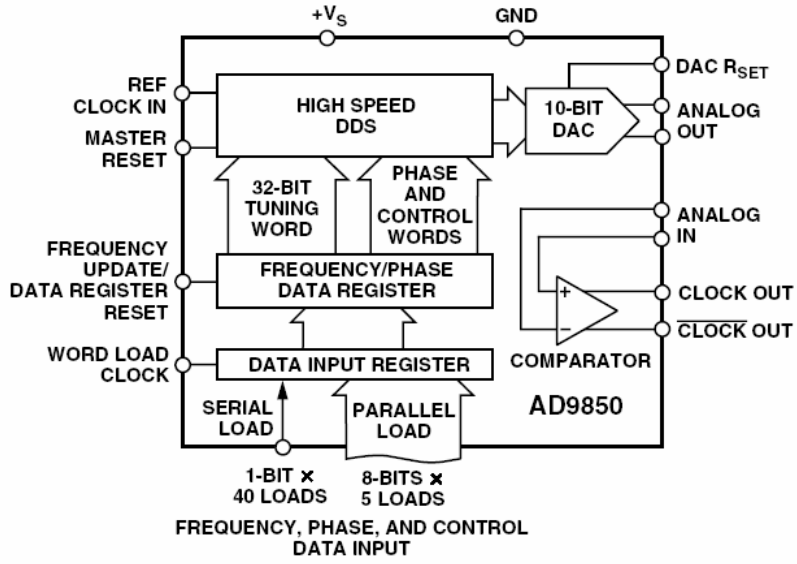


Bild 4.11: Blockschaltbild des DDS vom Typ AD9850

Wie das Blockschaltbild zeigt, verfügt er nur über die wesentlichen Funktionen zur Frequenzerzeugung. Darüber hinaus ist lediglich eine eingeschränkte Phasenmodulation mit 5bit möglich, d.h. mit einer Auflösung von 11,25°. Die Programmierung kann wieder entweder seriell oder parallel geschehen, letztere aber nur über einen 8bit-BUS. Eine Adressierung interner Register ist nicht nötig, da mit jeder Eingabe stets alle 40 bit (5·8bit) beschrieben werden.

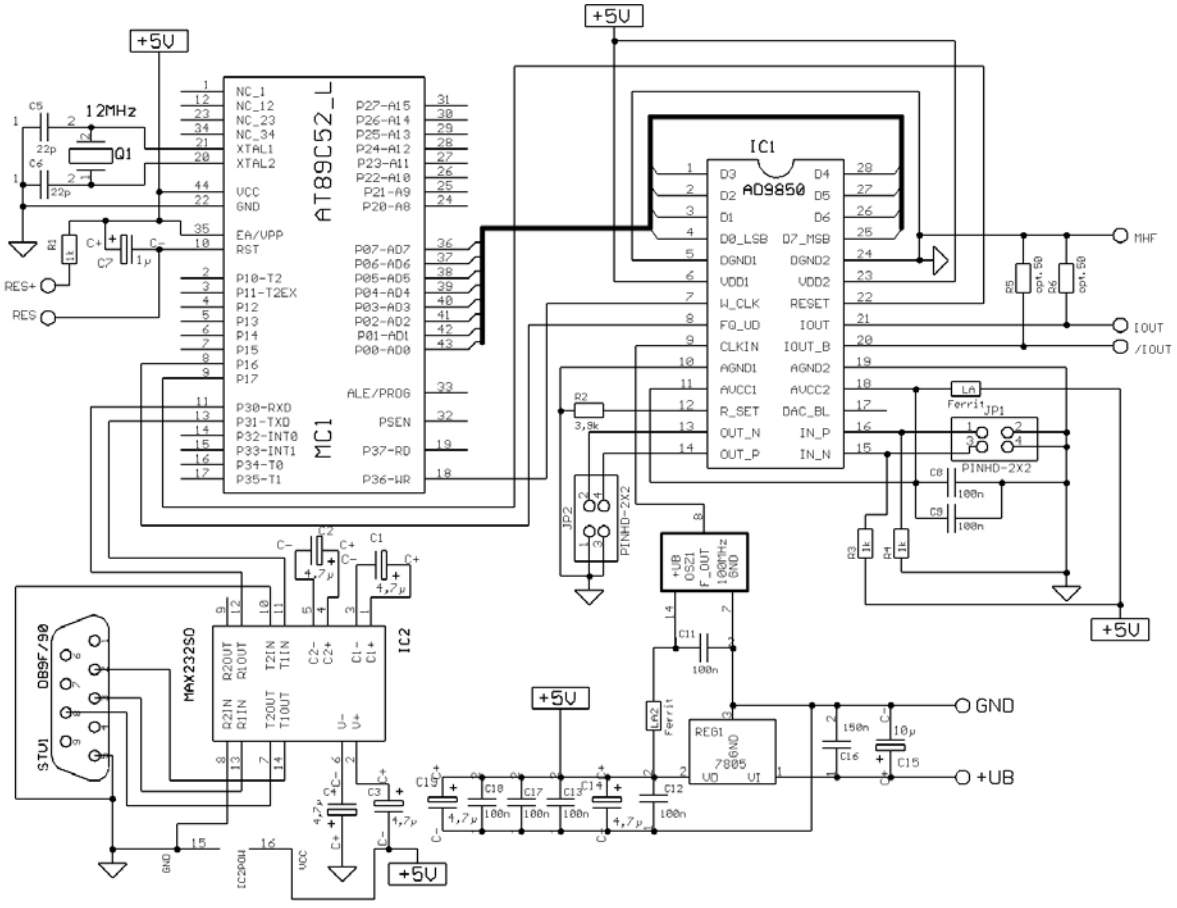


Bild 4.12: Anschluß eines AD9850 an einen 8bit-Mikrocontroller

Wie Bild 4.12 zeigt, ist auch dieser DSS sehr einfach mit einem Mikrocontroller vom Typ 8052 zu verbinden. Neben direkt im Mikrocontroller verfügbaren Routinen zur Frequenzerzeugung erlaubt auch hier die serielle Schnittstelle des MC wieder eine flexible und übersichtliche Programmierung von einem PC oder Notebook aus.

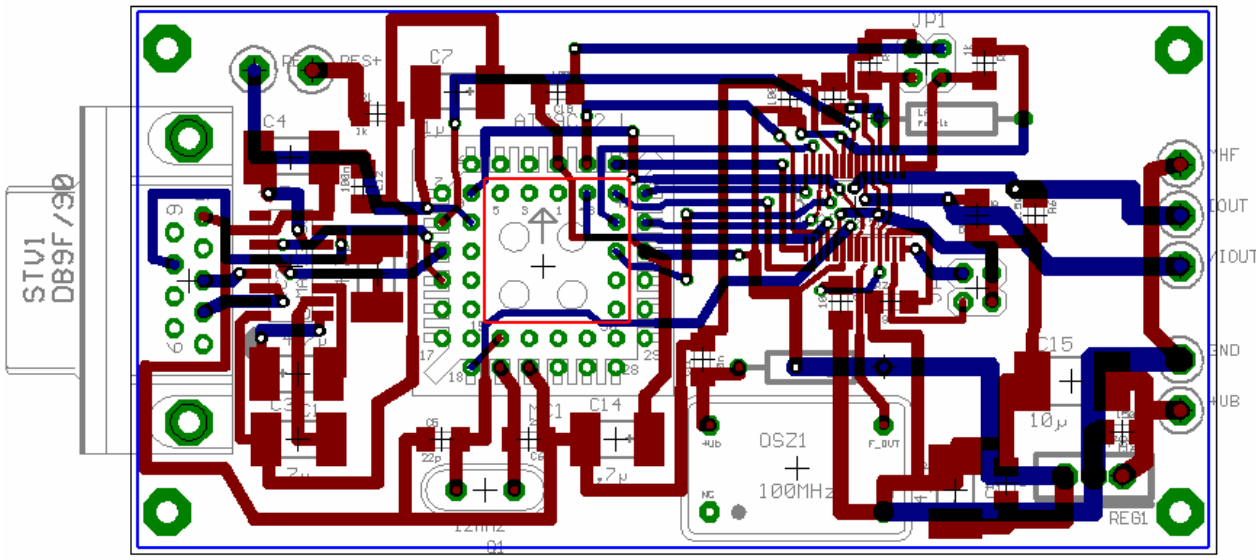


Bild 4.13: Platinenlayout für den Betrieb eines AD9850 mit einem 80C52-Mikrocontroller

4.2.1 Fehleruntersuchung bei der digitalen Signalsynthese über D/A-Wandler

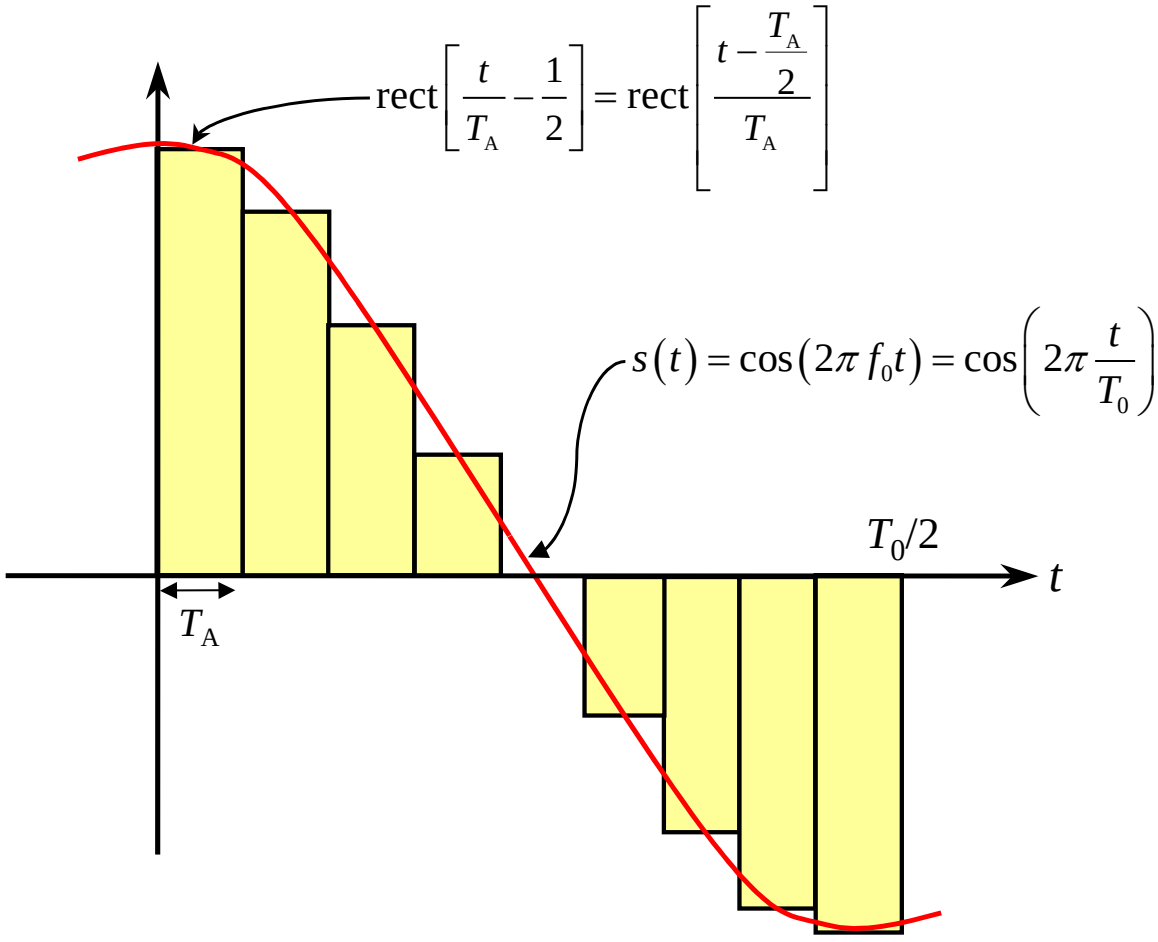


Bild 4.14: Der reale D/A-Wandler liefert Rechtecksignale am Ausgang

Anhand von (4.14) und Bild 4.4 wurde bereits gezeigt, daß die nichtideale Repräsentation von Abtastwerten, d.h. das Ausgeben einer Treppenfunktion anstelle von Diracimpulsen, zu einer Amplitudenverzerrung führt. Wenn es darum geht einzelne Spektrallinien zu generieren, kann auf relativ einfache Weise eine Korrektur vorgenommen werden – s. (4.15). Schwieriger wird es jedoch, wenn große Frequenzbereiche quasikontinuierlich zu überstreichen sind. Wegen der hohen praktischen Bedeutung dieser Thematik wird sie anhand von Bild 4.14 nochmals aufgegriffen und an Anwendungsbeispielen des AD9850 vertieft.

Aus (3.12) und (3.14) ist bekannt, daß für Abtastfolgen die Fourierbeziehung

$$\sum_{n=-\infty}^{\infty} \delta\left(t - nT_A\right) \circ\!\!-\!\!\bullet \frac{1}{\left|T_A\right|} \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{T_A}\right) \quad oder \frac{1}{T_A}\,\mathrm{III}\left(\frac{t}{T_A}\right) \circ\!\!-\!\!\bullet \mathrm{III}\left(fT_A\right) \tag{4.23}$$

gilt. Für die Fouriertransformierte eine des Cosinussignals gemäß Bild 4.14 erhält man

$$s(t) = \cos\left(2\pi f_0 t\right) \quad \circ\!\!-\!\!\bullet \quad \frac{1}{2}\left[\delta\left(f - f_0\right) + \delta\left(f + f_0\right)\right] \tag{4.24}$$

Würde ein solches Signal mit der idealen Folge nach (4.23) abgetastet, hätte man

$$s_{\mathrm{a}}(t) = s(t) \cdot \sum_{n=-\infty}^{\infty} \delta\left(t - nT_{\mathrm{A}}\right) \ . \tag{4.25}$$

Da aber einer Repräsentation der Abtastwerte in Form von Rechtecken der Dauer T_{A} vorliegt, ist eine entsprechen Faltung vorzunehmen, aus der man

$$s_{\mathrm{ra}}(t) = s(t) \cdot \sum_{n=-\infty}^{\infty} \delta\left(t - nT_{\mathrm{A}}\right) * \mathrm{rect}\left(\frac{t - \frac{T_{\mathrm{A}}}{2}}{T_{\mathrm{A}}}\right) \tag{4.26}$$

erhält. Die komplette Fouriertransformation liefert dann

$$\begin{aligned} S_{\mathrm{ra}}\left(f\right) &= \frac{1}{2}\left[\delta\left(f - f_0\right) + \delta\left(f + f_0\right)\right] * \frac{1}{T_A} \cdot \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{T_A}\right) \cdot T_A \cdot si\left(\pi\, f\, T_A\right) \cdot \mathrm{e}^{-j\pi f T_A} \\ &= \frac{1}{2} \cdot si\left(\pi\, f\, T_A\right) \cdot \mathrm{e}^{-j\pi f T_A} \cdot \left[\sum_{n=-\infty}^{\infty} \delta\left[f - \left(\frac{n}{T_A} + \frac{1}{T_0}\right)\right] + \sum_{n=-\infty}^{\infty} \delta\left[f - \left(\frac{n}{T_A} - \frac{1}{T_0}\right)\right]\right] \end{aligned} \tag{4.27}$$

Ein Beispiel mit T_0=3T_{A} soll die Zusammenhänge verdeutlichen:

$$S_{\mathrm{ra3}} = \frac{1}{2}\mathrm{si}\left(\pi\, f\, T_{\mathrm{A}}\right) \cdot \mathrm{e}^{-j\pi f T_{\mathrm{A}}} \left[\sum_{n=-\infty}^{\infty} \delta\left[\left(f - \frac{3n+1}{3T_{\mathrm{A}}}\right)\right] + \sum_{n=-\infty}^{\infty} \delta\left[\left(f - \frac{3n-1}{3T_{\mathrm{A}}}\right)\right]\right] \tag{4.28}$$

Betrachtet man zunächst das Grundspektrum, d.h. setzt man n=0 in (4.28), dann ergibt sich

$$S_0 = \frac{1}{2}\mathrm{si}\left(\pi\, f\, T_{\mathrm{A}}\right) \cdot \mathrm{e}^{-j\pi f T_{\mathrm{A}}} \left[\delta\left(f - \frac{1}{T_0}\right) + \delta\left(f + \frac{1}{T_0}\right)\right]. \tag{4.29}$$

Entsprechend für *n*=1

$$S_1 = \frac{1}{2}\mathrm{si}\left(\pi\, f\, T_\mathrm{A}\right)\cdot \mathrm{e}^{-j\pi\, f\, T_\mathrm{A}}\left[\delta\left(f-\frac{4}{T_0}\right)+\delta\left(f-\frac{2}{T_0}\right)\right]$$

und für n=–1:

$$S_{-1} = \frac{1}{2}\mathrm{si}\left(\pi\, f\, T_\mathrm{A}\right)\cdot \mathrm{e}^{-j\pi\, f\, T_\mathrm{A}}\left[\delta\left(f+\frac{2}{T_0}\right)+\delta\left(f+\frac{4}{T_0}\right)\right]$$

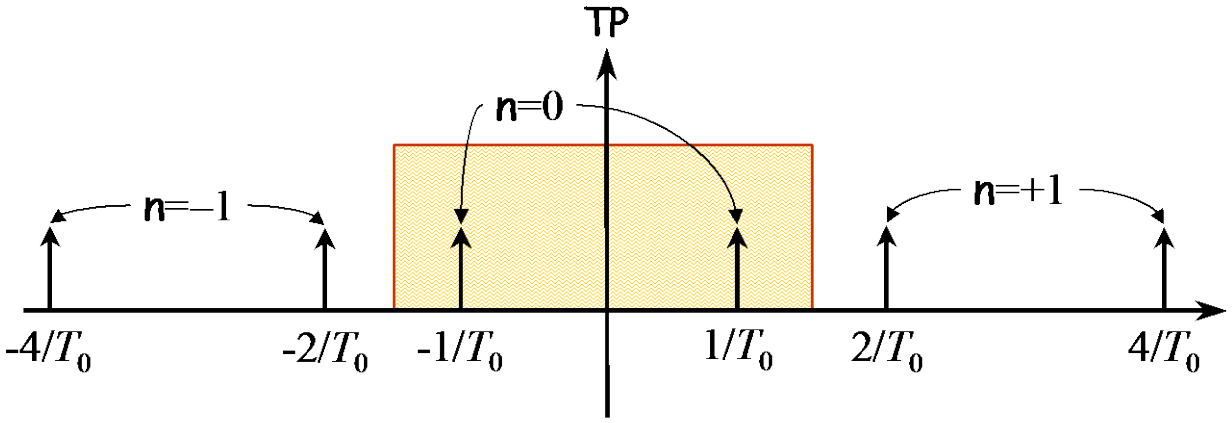


Bild 4.15: Gewünschtes Grundspektrum und die ersten ’Images’

Rekonstruktion des Grundspektrums: Tiefpaß mit $f_g < \frac{2}{T_0}$

Dann erhält man durch Ausmultiplizieren von (4.29) unter Berücksichtigung der ’Siebeigenschaft’ des Diracimpulses

$$S_0 = \frac{1}{2}\left[\mathrm{si}\left(\pi\frac{T_\mathrm{A}}{T_0}\right)\cdot \mathrm{e}^{-j\pi\frac{T_\mathrm{A}}{T_0}}\cdot\delta\left(f-\frac{1}{T_0}\right)+\mathrm{si}\left(\pi\frac{-T_\mathrm{A}}{T_0}\right)\cdot \mathrm{e}^{j\pi\frac{T_\mathrm{A}}{T_0}}\cdot\delta\left(f+\frac{1}{T_0}\right)\right]$$

$$= \frac{1}{2}\mathrm{si}\left(\pi\frac{T_\mathrm{A}}{T_0}\right)\left[\delta\left(f-\frac{1}{T_0}\right)\overbrace{\left[\cos\left(\pi\frac{-T_\mathrm{A}}{T_0}\right)+j\sin\left(\pi\frac{-T_\mathrm{A}}{T_0}\right)\right]}^{\mathrm{e}^{-j\pi\frac{T_\mathrm{A}}{T_0}}}+\delta\left(f+\frac{1}{T_0}\right)\overbrace{\left[\cos\left(\pi\frac{T_\mathrm{A}}{T_0}\right)+j\sin\left(\pi\frac{T_\mathrm{A}}{T_0}\right)\right]}^{\mathrm{e}^{j\pi\frac{T_\mathrm{A}}{T_0}}}\right]$$

$$= \frac{1}{2}\mathrm{si}\left(\pi\frac{T_\mathrm{A}}{T_0}\right)\cdot\left[\cos\left(\pi\frac{T_\mathrm{A}}{T_0}\right)\left[\delta\left(f-\frac{1}{T_0}\right)+\delta\left(f+\frac{1}{T_0}\right)\right]+\sin\left(\pi\frac{T_\mathrm{A}}{T_0}\right)\cdot j\left[\delta\left(f+\frac{1}{T_0}\right)-\delta\left(f-\frac{1}{T_0}\right)\right]\right]$$

Mittels Fourierrücktransformation folgt wegen:

$\cos(2\pi f_0 t) = \frac{1}{2}\left[\delta(f-f_0)+\delta(f+f_0)\right]$ und $\sin(2\pi f_0 t) = j\frac{1}{2}\left[\delta(f+f_0)-\delta(f-f_0)\right]$

$$s_0(t) = \mathrm{si}\left(\pi\frac{T_A}{T_0}\right)\cdot\left[\cos\left(\pi\frac{T_A}{T_0}\right)\cdot\cos\left(2\pi\frac{t}{T_0}\right)+\sin\left(\pi\frac{T_A}{T_0}\right)\cdot\sin\left(2\pi\frac{t}{T_0}\right)\right], \quad (4.30)$$

und unter Anwendung des Additionstheorems $\cos(\alpha-\beta)=\cos\alpha\cos\beta+\sin\alpha\sin\beta$ folgt schließlich das Endergebnis:

$$s_0(t) = \mathrm{si}\left(\pi \frac{T_A}{T_0}\right) \cdot \cos\left(2\pi f_0 t - \pi \frac{T_A}{T_0}\right) = \mathrm{si}\left(\pi \frac{f_0}{f_A}\right) \cdot \cos\left(2\pi f_0 t - \pi \frac{f_0}{f_A}\right) \qquad (4.31)$$

Amplitudenfaktor Phasenverschiebung

Das synthetisierte Analogsignal weist also abhängig vom Verhältnis T_A/T_0 bzw. f_0/f_A eine Amplitudengewichtung und eine Phasenverschiebung auf. Während die Phasenverschiebung in vielen Fällen keine Rolle spielt, kann der Amplitudenfaktor in der Regel nicht vernachlässigt werden, so daß eine entsprechende Korrektur benötigt wird – s. auch Bild 4.4 und (4.15). Bei der Nyquistrate wäre die Amplitude bereits auf weniger als 64% abgesunken, so daß schon lange vorher korrigiert werden muß. Weil aufgrund der hohen Frequenzauflösung beim DDS eine kontinuierliche Korrektur benötigt wird, ist der Einsatz eines entsprechenden Digitalfilters nahe liegend, das den $\mathrm{si}\left(\pi \, f / f_A\right)$-Verlauf „glattbügelt“. Solche Filter werden in der Literatur auch **sinc^{-1}**-Filter genannt und sind meist als FIR-Filter relativ hoher Ordnung realisiert. Moderne DDS–Bausteine wie z.B. die Typen AD9852/54 (Analog Devices) haben solche Filter integriert. Sie werden aber nur bei Bedarf zugeschaltet, da sie erheblich zum Anstieg der Leistungsaufnahme beitragen.

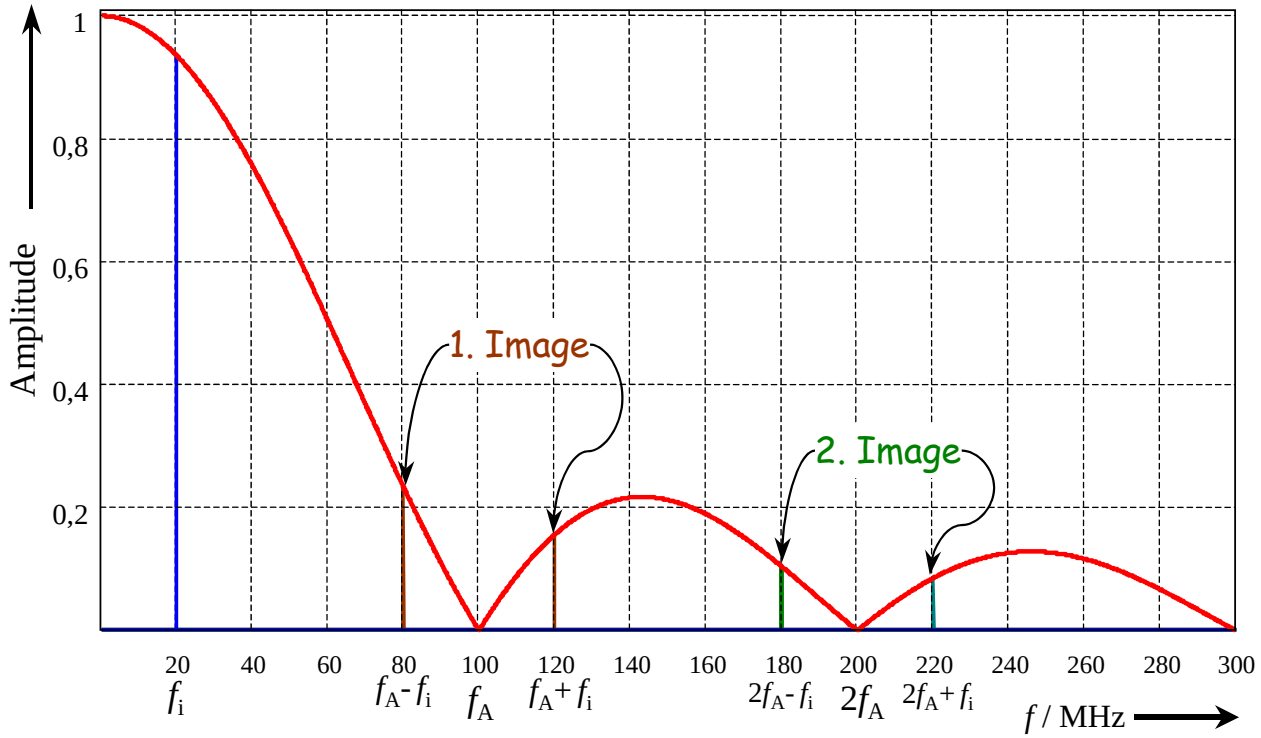


Bild 4.16: Amplitudengewichtung und „Images“ bei der Synthese von f_i= 20MHz mit einem AD9850, bei dem die Quarzreferenz f_A=100MHz ist

Wie man aus Bild 4.16 ersieht, weist die gewünschte Spektrallinie bei 20MHz einen Amplitudenabfall auf si(0,2·π)=0,93549 auf, während die erste „Spiegelfrequenz“ bei 80MHz auf si(0,8·π)=0,234 abgedämpft ist. Der Abstand von 60MHz erlaubt eine saubere Selektion der gewünschten Frequenz mit einfachen Mitteln.

Wenn sich die gewünschte Ausgangsfrequenz jedoch der Nyquistrate nähert, wird die Selektion erheblich aufwendiger, wie Bild 4.17 demonstriert. Zum einen ist weist die gewünschte Spektrallinie bei 45 MHz schon einen Amplitudenabfall auf si(0,45·π)=0,698 auf, während die erste „Spiegelfrequenz“ bei 55 MHz nur noch auf si(0,55·π)=0,571 abgedämpft ist. Der geringe Abstand von 10MHz erfordert jetzt hohen Aufwand für eine saubere Selektion der gewünschten Frequenz.

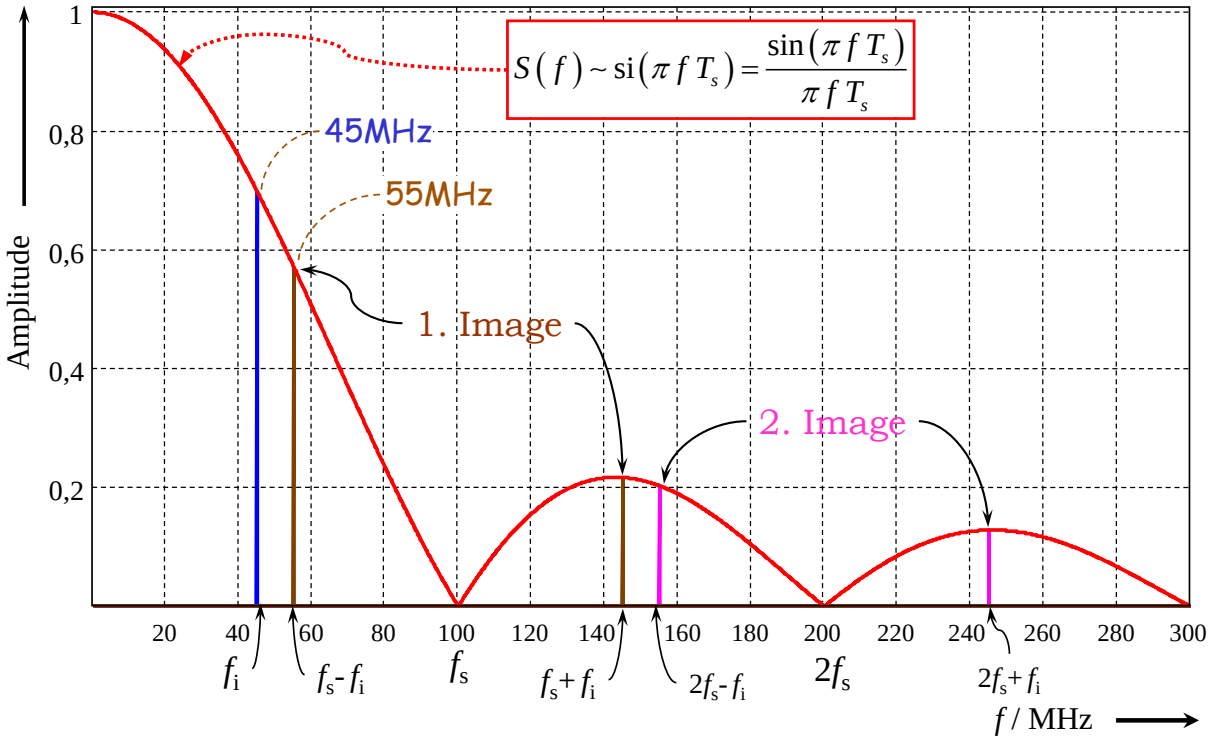


Bild 4.17: Amplitudengewichtung und „Images“ bei der Synthese von f_i= 45MHz mit einem AD9850, bei dem die Quarzreferenz f_A=100MHz ist

Wenn sich die gewünschte Ausgangsfrequenz noch weiter der Nyquistrate nähert, tritt ein interessantes Phänomen auf. Aufgrund der engen Nachbarschaft von gewünschter Ausgangsfrequenz und erster Spiegelfrequenz nehmen beide die gleiche Amplitude an und eine Trennung ist nicht mehr möglich. Somit erhält man unter Vernachlässigung bzw. nach Unterdrückung der höheren Spiegelfrequenzen die Summe von zwei eng benachbarten Spektrallinien, d.h. ein Signal der Form

$$g(t) = \cos(2\pi f_1 t) + \cos(2\pi f_2 t) = 2 \cdot \cos\left[2\pi \frac{f_1 + f_2}{2} t\right] \cdot \cos\left[2\pi \frac{f_1 - f_2}{2} t\right] \qquad (4.32)$$

unter Anwendung des Additionstheorems $\cos(\alpha) + \cos(\beta) = 2 \cdot \cos\left[\frac{\alpha + \beta}{2}\right] \cdot \cos\left[\frac{\alpha - \beta}{2}\right]$.

Der rechte Teil von (4.32) stellt ein Produkt aus einer hochfrequenten (f_1+f_2) und einer niederfrequenten (f_1–f_2) Schwingung dar. Dies beschreibt exakt die Amplitudenmodulation eines (hochfrequenten) Trägers mit einem niederfrequenten Nachrichtensignal. Ein einfaches Beispiel mit den Frequenzen f_1=98 Hz und f_2=102 Hz illustriert den Vorgang. Damit ergibt sich die ***'Trägerfrequenz'*** f_t =100 Hz und die ***'Modulationsfrequenz'*** f_m=2 Hz.

Wenn man den DDS–Baustein AD9850 mit f_A=100 MHz betreibt, läßt sich der beschriebene Effekt in der Nähe von 50 MHz beobachten. D.h. man könnte f_1=49.999.998 Hz einstellen. Dann wäre die 1. Spiegelfrequenz f_2=100 MHz–f_1 = 50.000.002 Hz. Gemäß (4.32) hätte man jetzt eine ***Trägerfrequenz*** f_t =50 MHz und die ***Modulationsfrequenz*** wäre wieder f_m=2 Hz. Wegen der hohen Frequenzauflösung ($f_{min} = \frac{100\mathrm{MHz}}{2^{32}} \approx 23\mathrm{mHz}$) läßt sich der Effekt noch bis zu einer Modulationsfrequenz von 23 mHz weitertreiben, so daß eine Periode rund 43 s dauert.

4.2.2 Lineare Frequenzmodulation (***Chirp–Signal***)

$$s_{CH}(t) = A \cdot rect(t/T) \cdot \cos\left[2\pi f_0 t + 1/2\,\mu t^2\right] \tag{4.33}$$

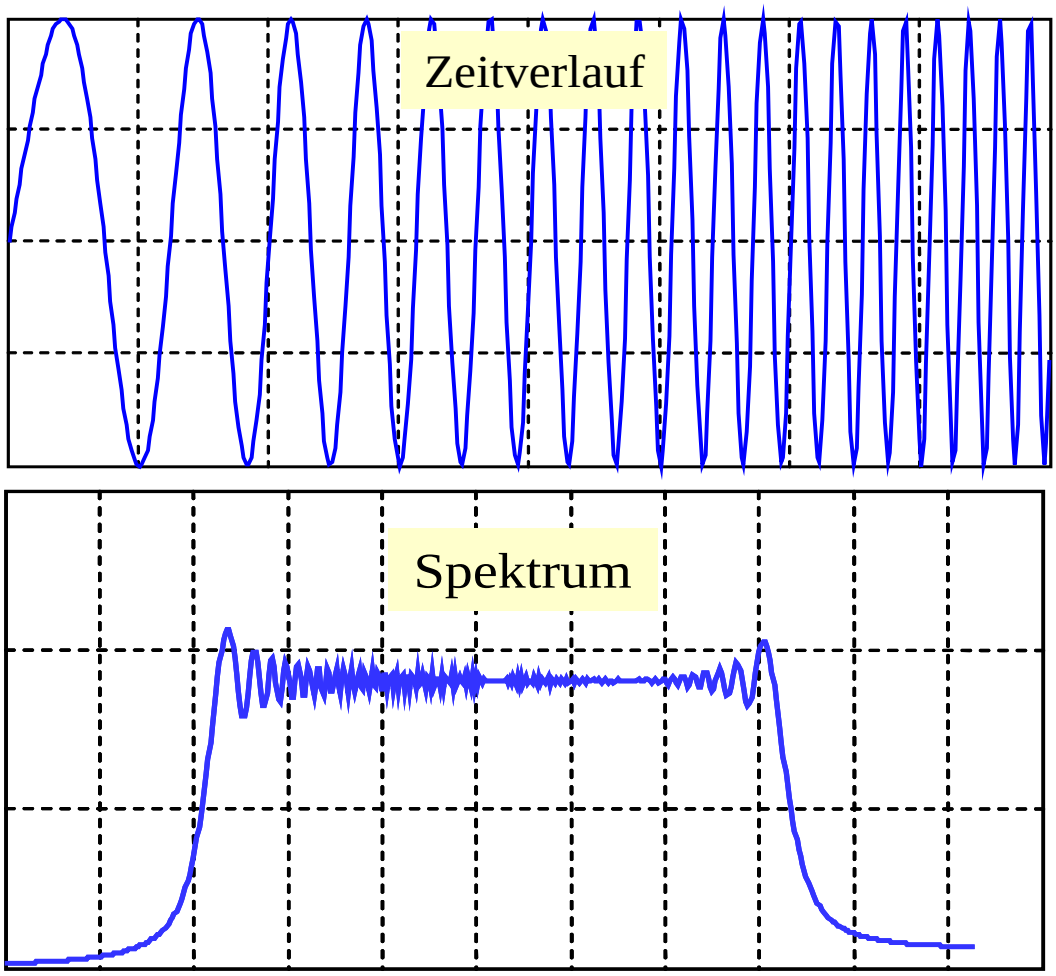


Bild 4.18: Zeitverlauf und Spektrum eines Chirp–Signals (lineare Frequenzmodulation)

(4.33) beschreibt ein cosinusförmiges Signal, dessen Frequenz über eine bestimmte Dauer *T* linear mit der Zeit zunimmt. Durch Ableitung des Arguments des Cosinus kann eine Momentanfrequenz angegeben werden:

$$f_{iCH}(t) = \frac{1}{2\pi}\frac{d\psi(t)}{dt} = f_0 + \frac{1}{2} \cdot 2\mu \cdot t = f_0 + \mu \cdot t \tag{4.34}$$

angegeben werden. Wie man sieht wächst die Frequenz linear mit der Zeit. Solche Chirpsignale haben sowohl in der Meß- als auch in der Kommunikationstechnik eine Bedeutung. Sie zählen zur Klasse der Bandspreizsignale (engl.: spread spectrum waveforms) und zeichnen sich durch die besonders kompakte Form des Spektrums aus – s. Bild 4.18. Die Verwendung von Bandspreizsignalen in der Kommunikationstechnik wird später behandelt; dabei wird auch die besondere Rolle der Chirp–Signale weiter verdeutlicht.

Zum Abschuß dieses Kapitels, das sich schwerpunktmäßig mit der digitalen Signalsynthese befaßte, kommen wir noch mal auf das DDS–Prinzip und seine Erweiterung zum universellen numerisch gesteuerten Modulationssystem zurück, ohne das heutige komplexe digitale Modulationsver-

fahren wie QAM[7] nicht realisierbar wären. Bevor tiefer gehende theoretische Zusammenhänge näher behandelt werden, erfolgt zunächst eine anschauliche Darstellung von QAM, die den direkten Bezug zur DDS- bzw. NCMO–Hardware herstellt. QAM fällt in die Klasse der ***polaren*** Modulation, was sicherlich nichts mit Kälte zu tun hat, sondern mit der graphischen (vektoriellen) Darstellung, die nun betrachtet wird.

Ein informationstragendes Sendesignal, das entweder in ein Kabelsystem oder auf eine Antenne gespeist wird, kann allgemein in der Form

$$\begin{aligned} s(t) &= \mathrm{Re}\left\{\underline{u}(t)\cdot \mathrm{e}^{\mathrm{j}2\pi f_0 t}\right\} = \mathrm{Re}\left\{\left(I_{\underline{u}} + \mathrm{j}Q_{\underline{u}}\right)\cdot\left[\cos(2\pi f_0 t) + \mathrm{j}\sin(2\pi f_0 t)\right]\right\} = \\ &= I_{\underline{u}}\cdot\cos(2\pi f_0 t) - Q_{\underline{u}}\cdot\sin(2\pi f_0 t) \end{aligned} \tag{4.35}$$

Wie man sieht, ist *s*(*t*) ein reelles Signal, was eine Grundvoraussetzung dafür ist, daß es physikalisch existiert, d.h. mit geeigneten elektronischen Komponenten erzeugt werden kann. Das komplexe Signal $\underline{u}(t)$ trägt die Nachricht, die dem komplexen Träger aufmoduliert wird. In (4.35) wurde $\underline{u}(t)$ bereits in zwei (reelle) Bestandteile $I_{\underline{u}}$ und $Q_{\underline{u}}$, die Inphase- und Quadraturkomponente genannt werden, zerlegt. Mit der rein formalen Schreibweise

$$\underline{u}(t) = r(t)\cdot \mathrm{e}^{j\varphi t} = r(t)\cdot\cos\left[\varphi(t)\right] + \mathrm{j}\cdot r(t)\cdot\sin\left[\varphi(t)\right] \tag{4.36}$$

erhält man die schon erwähnte ***polaren*** Darstellung, wobei *r*(*t*) einen radialen Vektor darstellt, der nach Maßgabe des Winkels φ in der komplexen Ebene „liegt“. Ein Vergleich von (4.35) und (4.36) liefert

$$I_{\underline{u}} = r(t)\cdot\cos\left[\varphi(t)\right] \quad \text{und} \quad Q_{\underline{u}} = r(t)\cdot\sin\left[\varphi(t)\right], \tag{4.37}$$

womit sich das reelle Sendsignal jetzt als

$$s(t) = r(t)\cdot\left[\cos\left[\varphi(t)\right]\cdot\cos(2\pi f_0 t) - \sin\left[\varphi(t)\right]\cdot\sin(2\pi f_0 t)\right] \tag{4.38}$$

darstellen läßt. Unter Anwendung des Additionstheorems $\cos(\alpha+\beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$ erhält man das einfache Ergebnis

$$s(t) = r(t)\cdot\cos\left[2\pi f_0 t + \varphi(t)\right], \tag{4.39}$$

das besagt, daß die zu übertragende Information jetzt sowohl in der zeitlich veränderlichen Amplitude *r*(*t*) des Trägers mit der Frequenz f_0, als auch in der Phase φ(*t*) enthalten ist. Wie schon anhand von Bild 4.9 angedeutet, kann ein Signal gemäß (4.39) mit einem DDS–Baustein vom Typ AD7008 generiert werden. Der Begriff ***polare*** Datenmodulation und deren vollständige digitale Implementierung wird im folgenden Bild 4.19noch mal verdeutlicht. Wie man sieht, übernimmt der Baustein sowohl die Trägererzeugung als auch das Aufbringen der Phaseninformation mittels eines Addierers und der Amplitudeninformation mittels des Multiplizierers nach dem Signalformspeicher (SIN/COS-ROM). Für sehr hohe Frequenzen, z.B. f_0 >1GHz, ist das Verfahren nach Bild 4.19 nicht mehr praktikabel. Man kann dann den in Bild 4.9 angedeuteten Weg gehen, allerdings mit analoger Mischung am Senderausgang. D.h. die Komponenten $I_{\underline{u}}$ und $Q_{\underline{u}}$ werden zunächst digital erzeugt und dann in der Weise auf den Träger gebracht wie es Bild 4.20 darstellt.

[7] Quadratur-Amplitudenmodulation

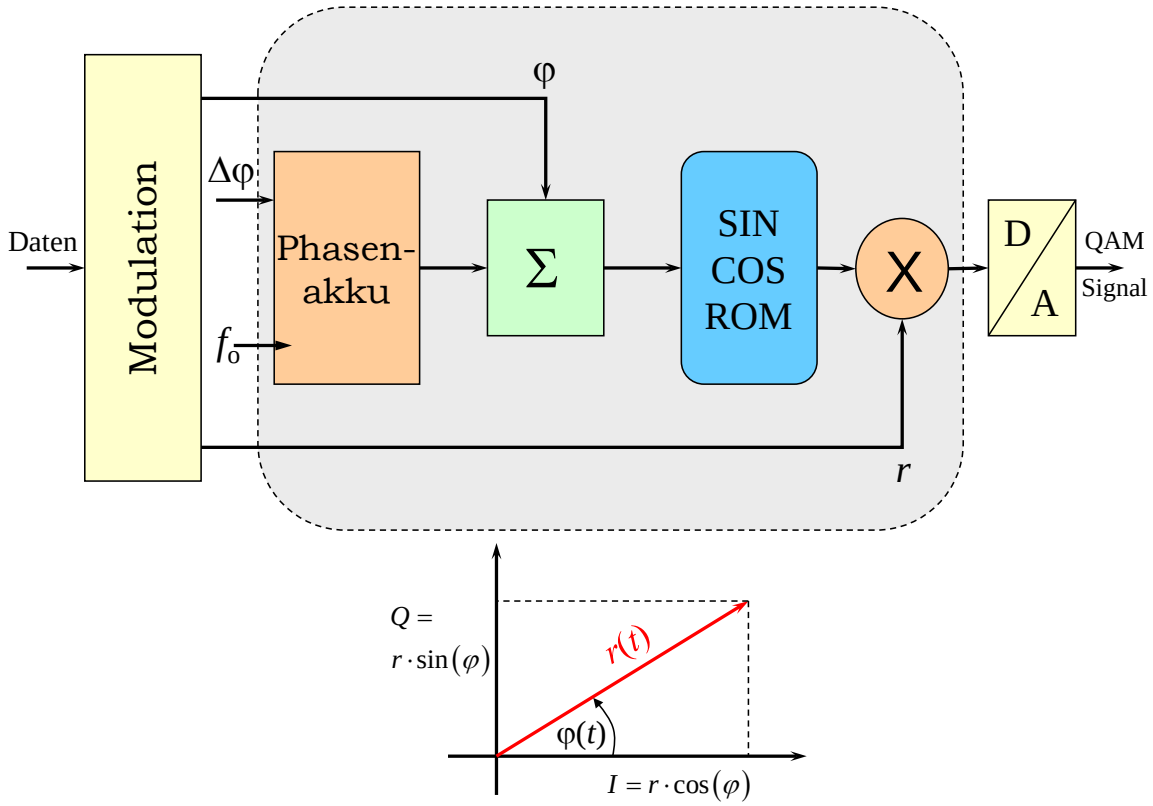


Bild 4.19: Numerisch gesteuerter Oszillator als polarer Datenmodulator

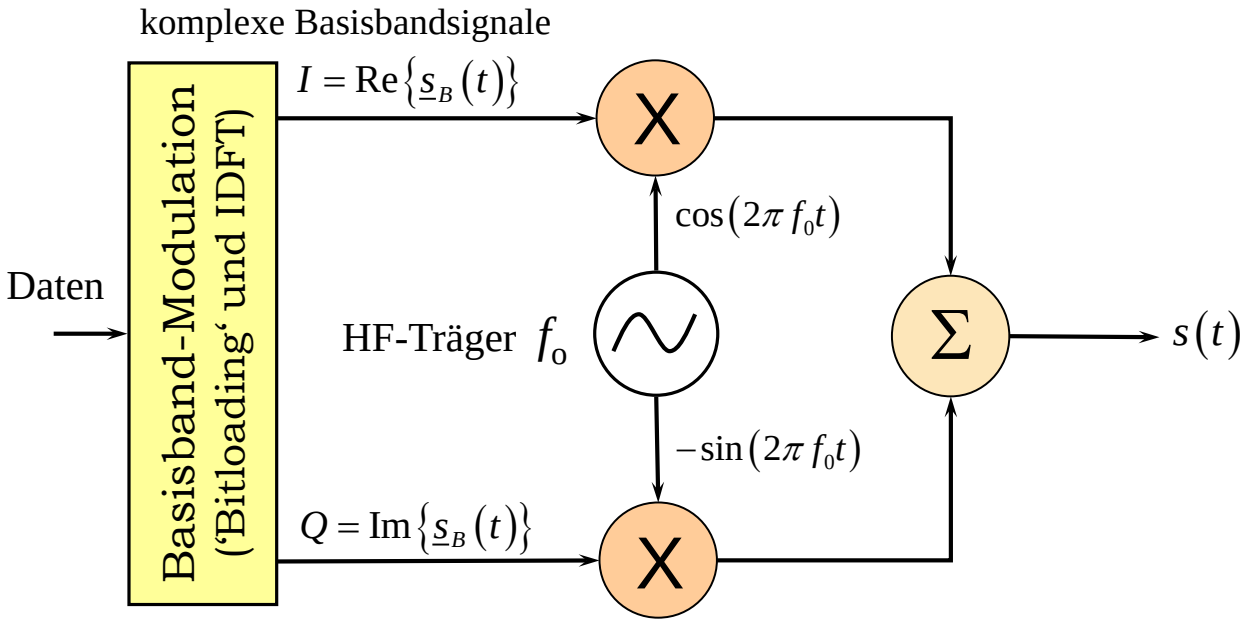


Bild 4.20: Polare Datenmodulation bei hohen Sendefrequenzen

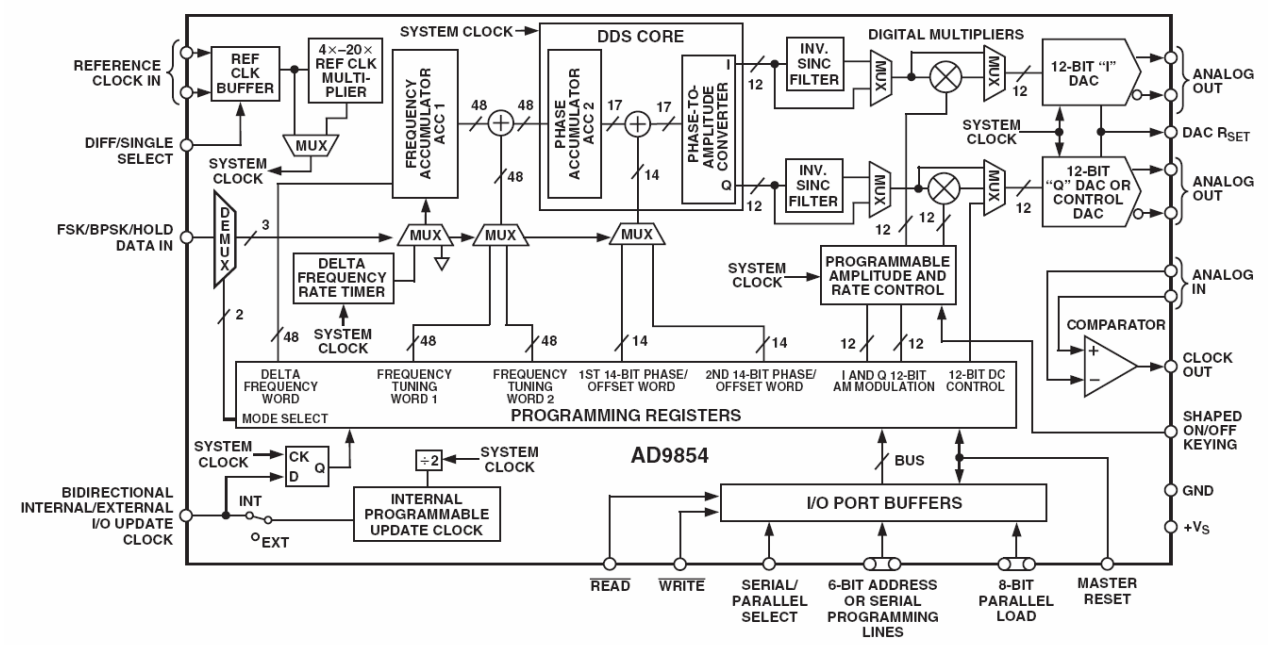


Bild 4.21: Blockschaltbild des „Quadratur–DDS“ AD9854

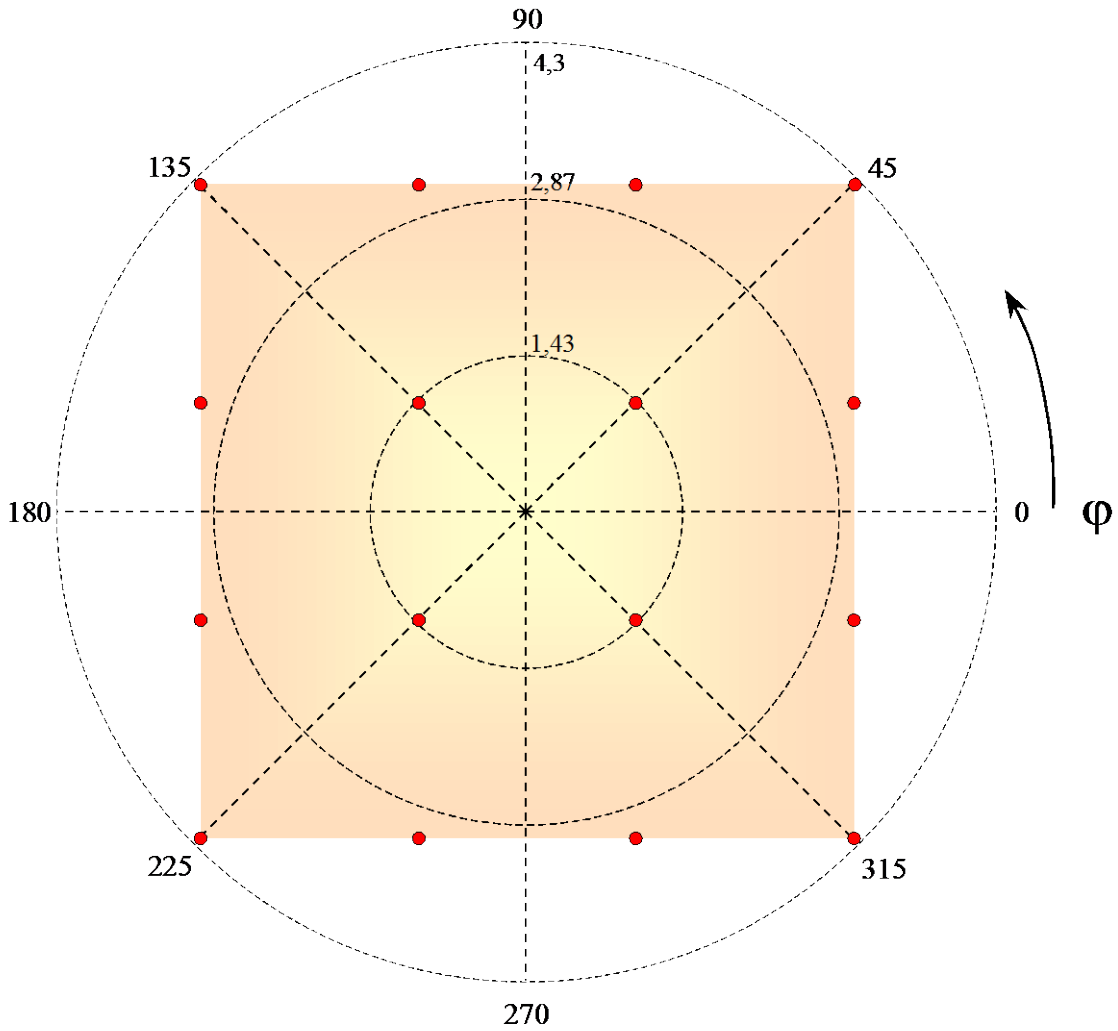


Bild 4.22: Polares Konstellationsdiagramm für 16-QAM

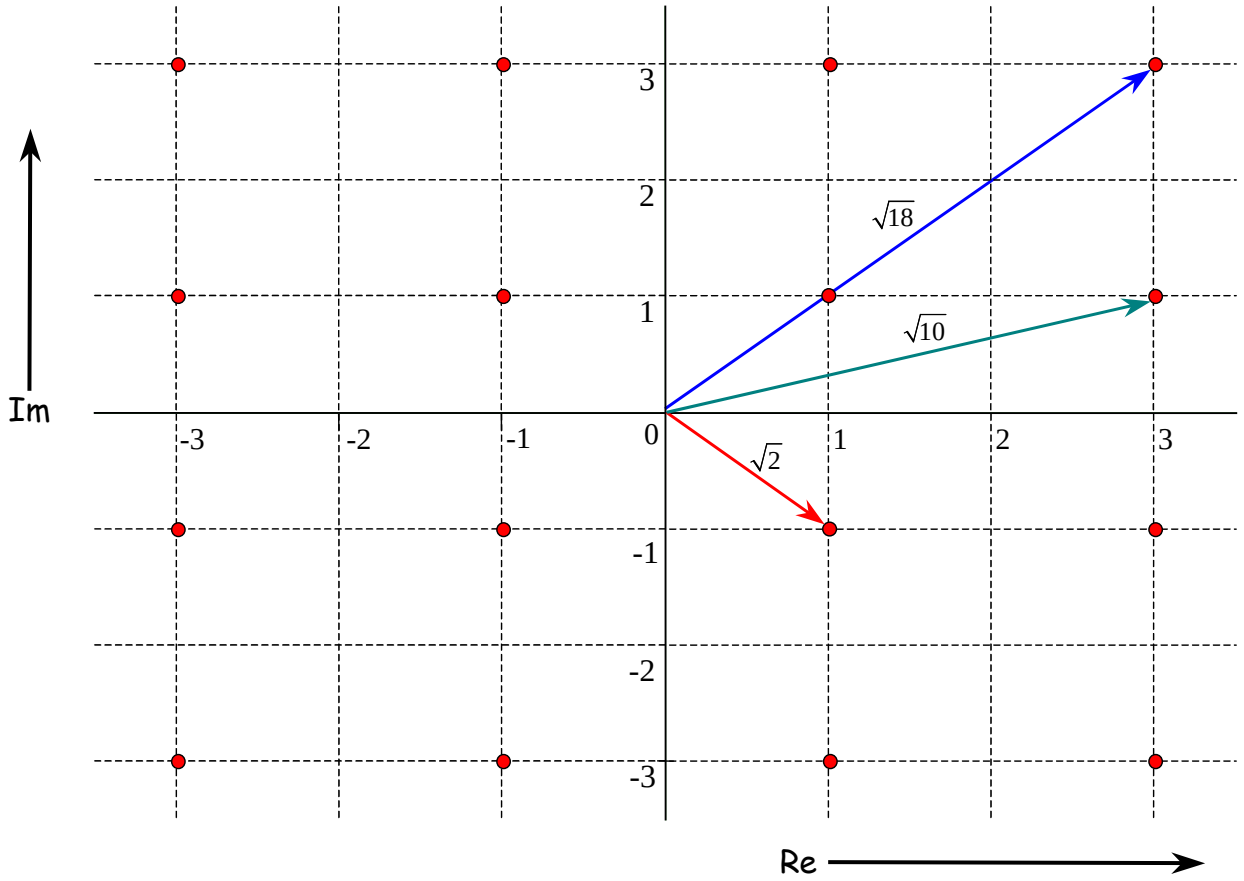


Bild 4.23: Die drei Amplitudenstufen bei 16-QAM

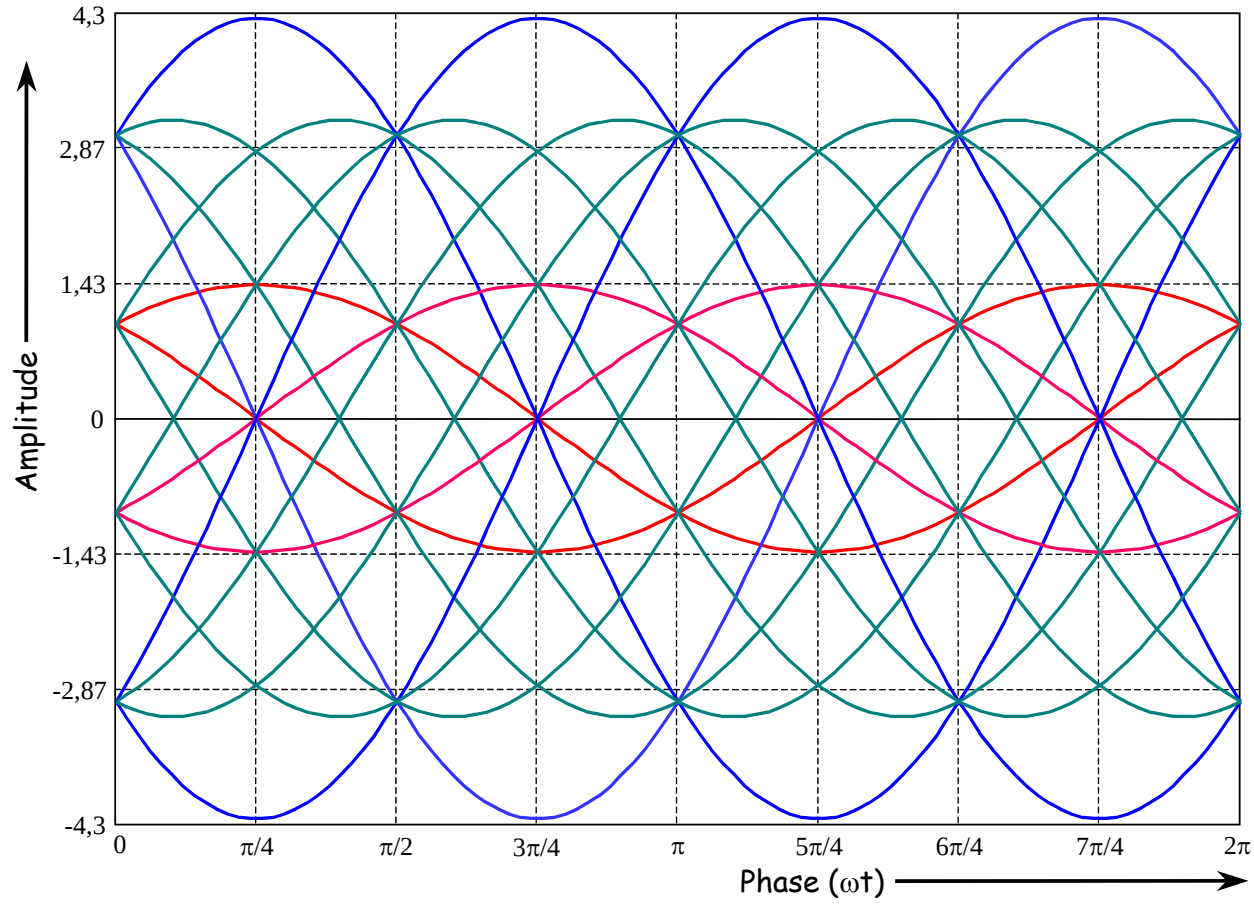


Bild 4.24: Die charakteristischen Zeitverläufe bei 16-QAM

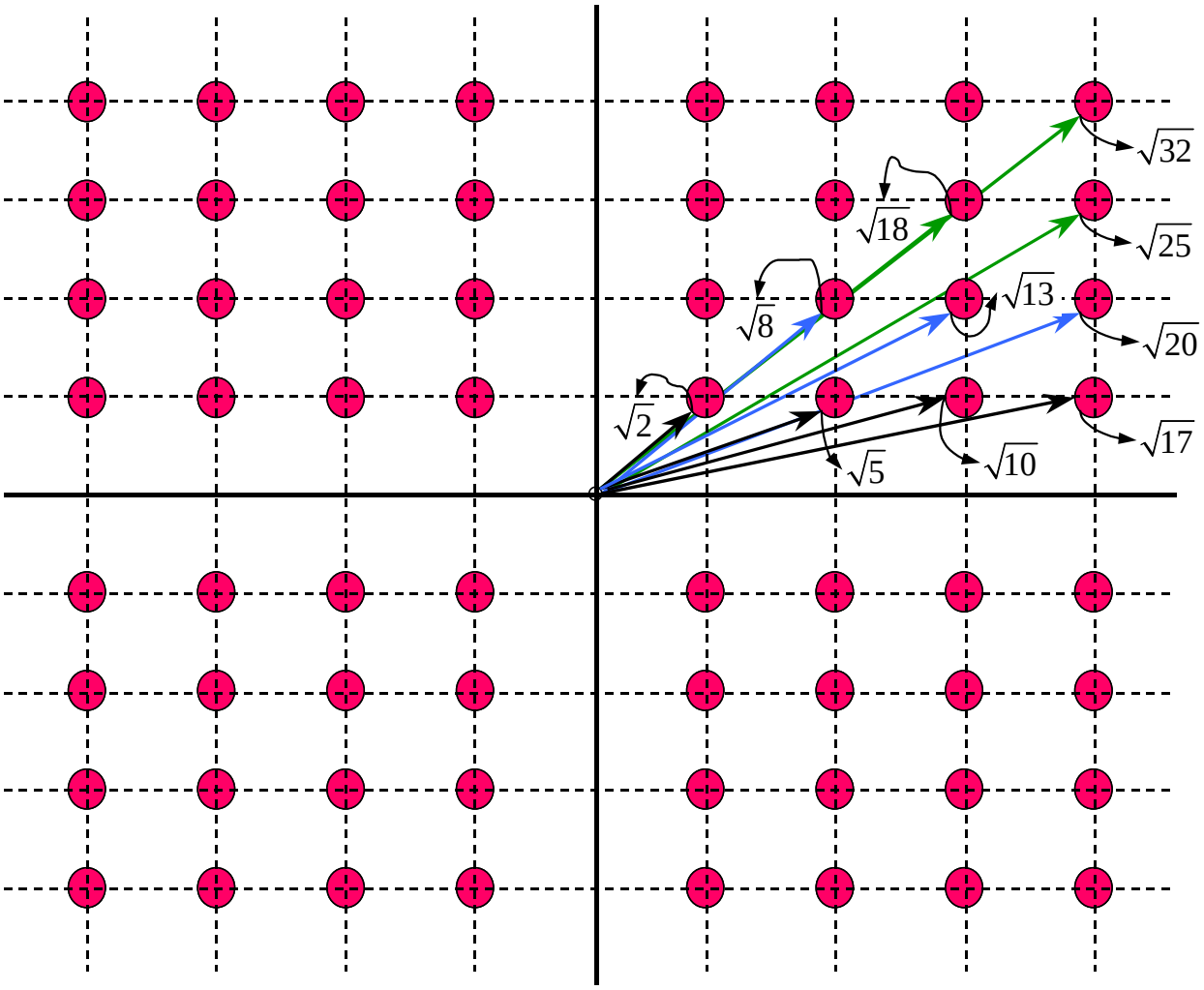


Bild 4.25: Konstellationsdiagramm für 64-QAM mit 10 Amplitudenstufen

4.2.2.1 Einfache digitale Modulationsverfahren

ASK: *Amplitude Shift Keying*

Ausgangspunkt ist sinusförmiges Trägersignal, $s(t) = A(t) \cdot \sin\left[2\pi f(t) \cdot t + \varphi(t)\right]$, das durch die drei Parameter Amplitude $\boldsymbol{A}$, Frequenz $\boldsymbol{f}$ und Nullphasenwinkel $\boldsymbol{\varphi}$ beschrieben wird.

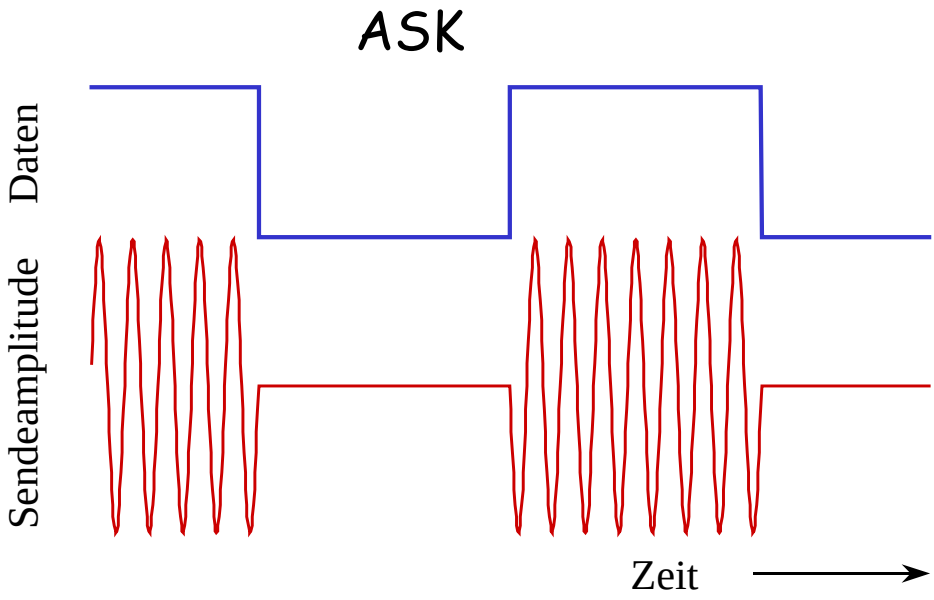
Auch bei der digitalen Amplitudenmodulation (ASK) findet eine zeitliche Veränderung der Trägeramplitude $A(t)$ nach Maßgabe eines digitalen Datenstroms statt. Im einfachsten Fall gibt es nur die beiden Werte Null und $\boldsymbol{A}$. D.h. bei ASK schaltet z.B. ein Null-Datenbit den Träger aus, während ein Eins-Datenbit ihn einschaltet. Man spricht daher auch von Amplitudentastung. Beschreibt man den zu übertragenden digitalen Datenstrom mit

$$d(t) = \sum_{i=-\infty}^{\infty} b_i \cdot rect\left(\frac{t - iT_b}{T_b}\right), \tag{4.40}$$

wobei b_i der binäre Datenvektor und T_b die Bitdauer ist, dann läßt sich ein ASK-Signal wie folgt darstellen:

$$s_{ASK}(t) = d(t) \cdot A \cdot \sin(2\pi\, f_0\, t) \tag{4.41}$$

ASK ist ein „historisches" Verfahren für die digitale Datenübertragung. Da Modulation und Demodulation technisch sehr einfach sind, konnte ASK auch ohne „Elektronik" realisiert werden.



Der entscheidende Nachteil bei ASK ist ihre hohe Störanfälligkeit. Denn jede Art additiver Störung wirkt sich unmittelbar auf die Empfangsamplitude aus, die die Information enthält. So bewirken additive Störungen, daß der Empfänger z.B. bei Null-Bits, wenn der Träger abgeschaltet ist, stets ein mehr oder weniger großes Signal erhält.

Bild 4.26: Daten- und Sendesignal bei ASK

Schon bei moderaten Dämpfungen des Nutzsignals kann es dann vorkommen, daß keine klare Trennung von Null- und Eins-Bit mehr möglich ist.

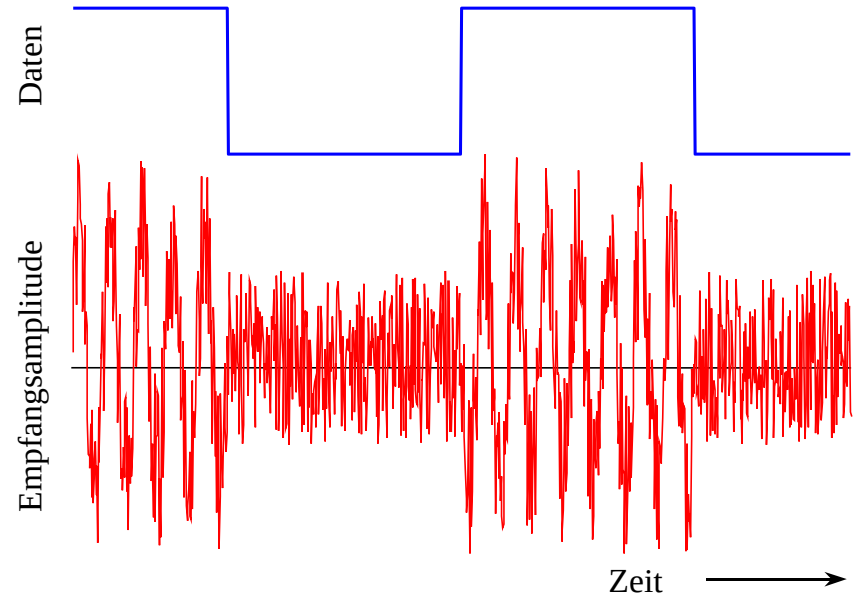
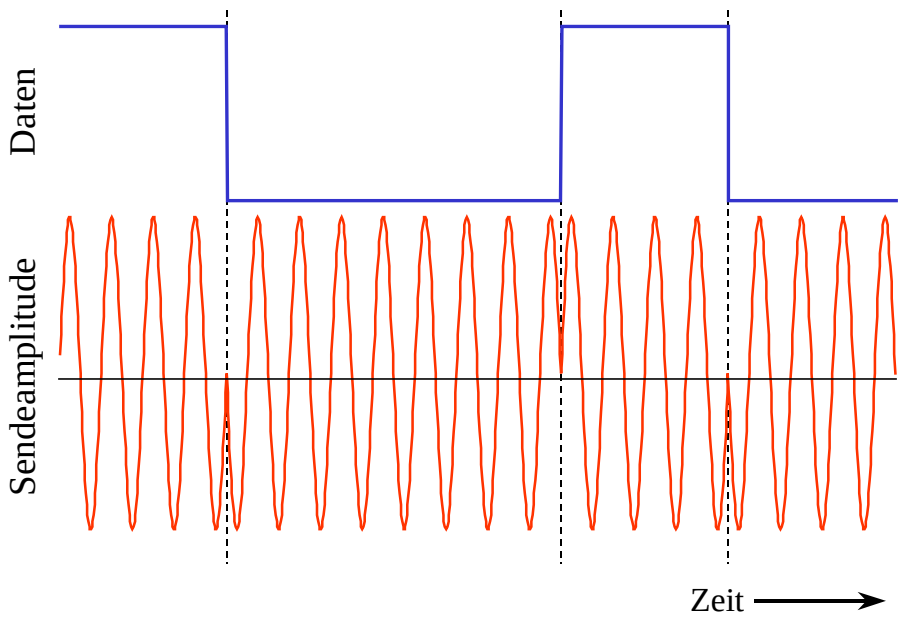


Bild 4.27 illustriert die Empfangssituation für das Signalbeispiel aus Bild 4.26, für den Fall, daß additive Störungen vorhanden sind. Man erkennt, daß der Abstand der Amplituden bei Null- und Eins-Bit deutlich verringert ist. Im dargestellten Fall wäre eine fehlerfreie Detektion der Datenbits zwar noch möglich, aber die Bitgrenzen würden schon einen beträchtlichen „Jitter" aufweisen.

Bild 4.27: ASK-Empfang mit additiver Störung

BPSK: *Binary Phase Shift Keying*

Wesentlich bessere Resultate erzielt man mit Phasenmodulation. Besonders einfach, aber dennoch höchst effizient im Hinblick auf Störfestigkeit ist die binäre Phasensprungmodulation - BPSK (Binary Phase Shift Keying). Hierbei wird der Nullphasenwinkel des Trägers entsprechend den logischen Zuständen eines binären Datensignals zwischen 0° und 180° umgeschaltet. Das läßt sich technisch leicht bewerkstelligen, so daß der Senderaufbau sehr einfach wird. Leider ist dies beim Empfänger nicht der Fall, weil bei der Demodulation eine exakte Kenntnis der Trägerphase benötigt wird. BPSK zeigt geringe Empfindlichkeit sowohl gegen Amplituden- als auch gegen Frequenzbeeinflussung. In der Klasse der schmalbandigen digitalen Modulationsverfahren ist BPSK darüber hinaus am wenigsten empfindlich gegen breitbandige Rauschstörer.



Der Grund für die allgemein hohe Störfestigkeit von BPSK ist der große ***euklidsche*** Abstand von $2 \cdot A$ ($A \equiv$Amplitude) der beiden Zustände im Signalraum. BPSK könnte dann empfindlich gestört werden, wenn dem Nutzsignal auf dem Weg vom Sender zum Empfänger weitere Phasensprünge zugesetzt würden. Dies ist jedoch in gewöhnlichen Störszenarien äußerst unwahrscheinlich.

Bild 4.28: Daten- und Sendesignalbeispiel für BPSK

Die Beschreibung für ein BPSK–Signal lautet:

$$s_{BPSK}(t) = A \cdot \sin\left[2\pi\, f_0\, t + d(t) \cdot \pi\right] \tag{4.42}$$

In Bild 4.28 sind im Sendesignal die 180° Phasensprünge an den Bitgrenzen deutlich zu erkennen. BPSK ist eine trägerunterdrückende Modulation, d.h. die gesamte Sendeleistung steht den informationstragenden Spektralanteilen zur Verfügung.

FSK: *Frequency Shift Keying*

Bei einfachen digitalen Datenübertragungssystemen wird der FSK meist der Vorzug gegenüber BPSK gegeben. Ein Hauptgrund dafür ist, daß FSK–Sende- und -Empfangssysteme sehr einfach sowohl in analoger als auch in digitaler Technik realisiert werden können. Während in den 70-er Jahren und Anfang der 80-er Jahre noch Analogtechnik vorherrschte, kam mit Beginn der 90er Jahre zunehmend digitale Technik ins Spiel. Anfangs wurde auf der Sendeseite einfach zwischen zwei Oszillatoren nach Maßgabe der zu sendenden Daten umgeschaltet, und auf Empfängerseite verwendete man häufig PLLs[8] zur Signaldetektion. In einem ersten Schritt der Digitalisierung wurden die Sender in Form digitaler Frequenzsynthesizer verschiedener Ausführung realisiert, während in die Empfänger nach und nach digitale Korrelatoren Einzug hielten. Heute sind zahlreich Varianten kompletter monolithisch integrierter FSK-Modems erhältlich, die auch die noch benötigten analogen Komponenten weitgehend einschließen.

Bild 4.29 zeigt ein FSK-Signalbeispiel. Bei FSK wird entsprechend einer logischen Null oder Eins im Datenstrom die Frequenz des Trägers sprungartig geändert. Für Null wird im Beispiel eine tiefe Frequenz und für Eins eine hohe Frequenz gesendet. Der Abstand der beiden Frequenzen ist in weiten Grenzen frei wählbar. Die untere Grenze für den Frequenzabstand ist der Wert der Datenrate, d.h. bei 4,8 kbit/s beispielsweise, müssen die beiden Frequenzen mindestens 4,8 kHz Abstand haben. Der Abstand darf aber auch ein ganzzahliges Vielfaches davon sein. Andere Frequenzabstände, die nicht in ganzzahligem Verhältnis zur Datenrate stehen, führen zum Verlust der ***Orthogonalität***.

[8] Phase Locked Loop

Das bedeutet, daß die Detektion im Empfänger bereits in moderater Störumgebung erheblich erschwert wird, weil sich ein „Übersprechen“ zwischen den Empfangszweigen für Null- und Eins-Bit einstellt. Wenn der Abstand der beiden FSK-Frequenzen exakt der Datenrate entspricht, erhält man die geringste Gesamtbandbreite des Sendesignals. Sie entspricht ungefähr der dreifachen Datenrate.

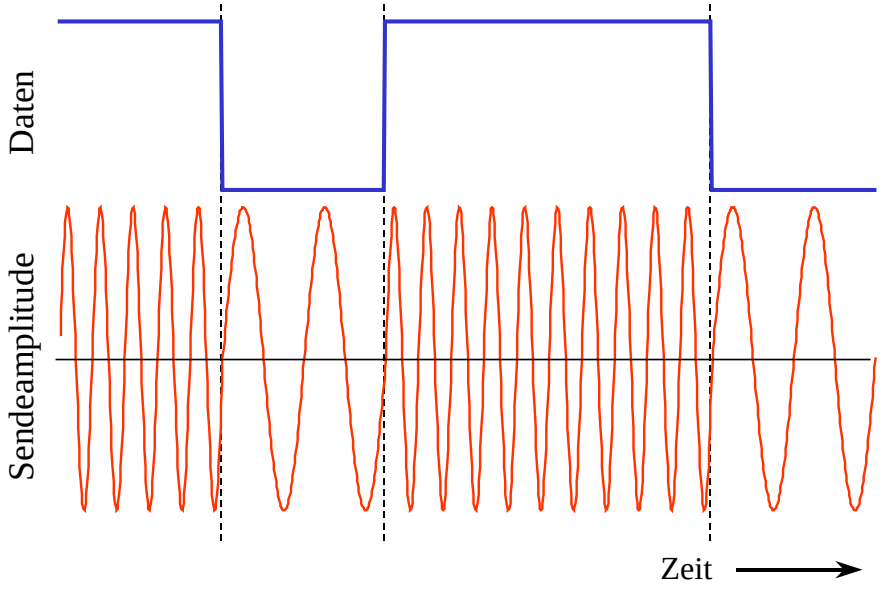


Bild 4.29: Daten- und Sendesignalbeispiel für FSK

FSK-Spektren für den Fall exakter Orthogonalität sind in Bild 4.30 dargestellt. Hier beträgt der Abstand der FSK-Frequenzen f_L=76,8kHz und f_H=81,6kHz genau 4800Hz, so daß dazu eine Datenrate von 4,8kbit/s paßt. Die gesamte Bandbreite von 14,4kHz entspricht etwa der dreifachen Datenrate. Entscheidend für die Orthogonalität ist, daß alle Nullstellen im Spektrum zusammenfallen. Bei dem Beispiel wird die Sendebandbreite relativ gut ausgefüllt, weil sich die um die Träger herum entstehenden Spektren, zur Hälfte überlappen. Wir werden auf diesen Sachverhalt bei der Betrachtung von OFDM[9] zurückkommen. Bei größerem Trägerabstand sinkt die spektrale Effizienz und es entstehen ungenutzte Lücken, die jedoch in gewissen Anwendungsfällen vorteilhaft oder sogar nötig sein können.

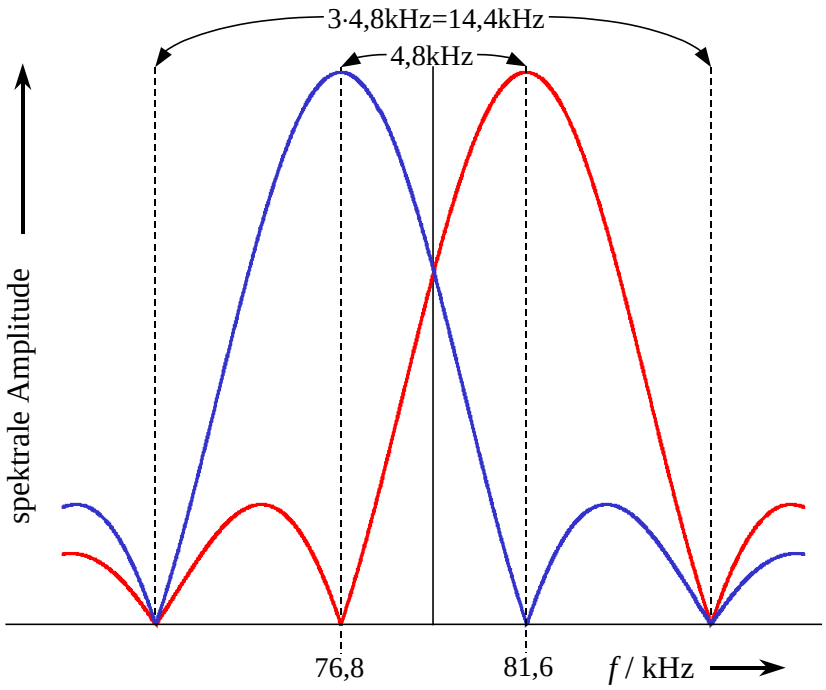
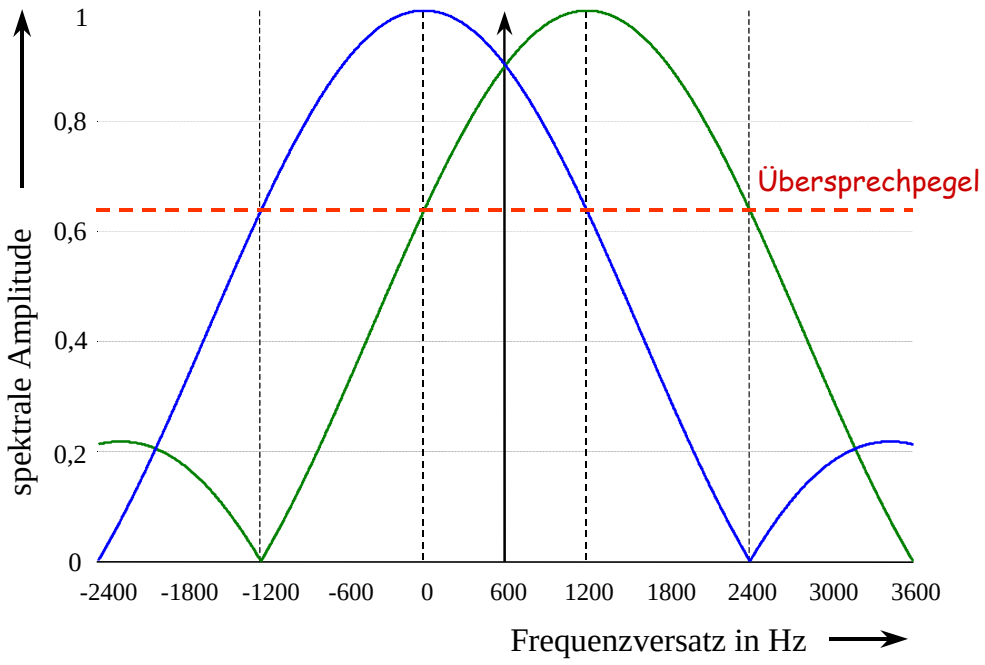


Bild 4.30: Beispiel zur Orthogonalität bei FSK

Für den Fall, daß der Trägerabstand weiter verkleinert wird, z.B. auf die Hälfte der Datenrate, tritt eine weitergehende Überlappung der Spektren auf, wobei das Zusammenfallen der Nullstellen nicht mehr gegeben ist.

9 Orthogonal Frequency Division Multiplexing



Ein solcher Fall ist in Bild 4.31 dargestellt. Hier ist bei einer Datenrate von 2400bit/s nur ein Trägerabstand von 1200Hz vorhanden. Wie man sieht, stellt sich an den Stellen der Maxima nicht mehr der Wert Null im jeweiligen anderen Empfangszweig ein, sondern ein beträchtlicher Pegel von rund 64%.

Bild 4.31: Beispiel zum Orthogonalitätsverlust bei zu kleinem Trägerabstand

Zur Vermeidung von Problemen der elektromagnetischen Verträglichkeit (EMV) durch ***Außerbandstörungen*** ist es vorteilhaft, wenn die FSK-Sendesignale phasenkontinuierlich sind, d.h. es darf keine Phasensprünge an den Bitübergängen geben - s. auch Bild 4.29. Aufgrund von Phasensprüngen entstehen breitbandige spektrale Anteile, die zwar für die gewünschte Übertragung kaum nachteilig sind, aber in benachbarten Frequenzbändern Kommunikationssysteme stören können. Phasenkontinuierliche FSK ist mit digitaler Synthese einfach zu realisieren.

FSK ist erheblich robuster als z.B. ASK, weil Verfälschungen der Frequenz durch überlagerte Störungen auf dem Übertragungsweg erheblich unwahrscheinlicher als Amplitudenverfälschungen sind. Bitfehler treten dann auf, wenn eine der beiden Frequenzen entweder extrem stark gedämpft von einer starken schmalbandigen Störung überlagert wird.

Ein FSK-Signal mit den Frequenzen f_L und f_H kann mit dem Datenvektor nach (4.40) wie folgt beschrieben werden:

$$s_{FSK}(t) = A \cdot \sum_{i=-\infty}^{\infty} \mathrm{rect}\left(\frac{t - iT_b}{T_b}\right) \cdot \left[b_i \cdot \sin\left[2\pi\, f_H\, t\right] + \overline{b}_i \cdot \sin\left[2\pi\, f_L\, t\right]\right] \qquad (4.43)$$

Bei b_i=1 wird also ein Träger der Frequenz f_H während der Bitdauer T_b gesendet und bei b_i=0 kommt die Frequenz f_L zum Zuge.

FH: *Frequency-Hopping*

FH ist die Grundlage für verschiedenartige Konzepte der störresistenten digitalen Datenübertragung. Es ist ein klassisches ***Spread-Spectrum***-Verfahren, weil kein Trägersignal fester Frequenz, sondern eine große Anzahl von Signalformen verschiedener Frequenz eingesetzt wird. Ein ausgesendetes FH-Signal wechselt in rascher Folge mit der Sprung- oder Hoprate h_r die Frequenz. Je kürzer die Verweildauer T_h=1/h_r auf einer Frequenz ist, um so weniger determiniert, d.h. um so rauschähnlicher erscheint einem Beobachter das FH-Signal. Die Verweildauer T_h wird im folgenden je nach Zusammenhang auch Frequenzgültigkeitsintervall oder „Chipdauer“ genannt. Ein „Chip“ eines FH-Signals mit der Amplitude A und der Momentanfrequenz f_m kann in der Form

$$s_{\mathrm{FH}}(t) = A \cdot \mathrm{rect}\left(\frac{t}{T_{\mathrm{h}}}\right) \cdot \sin(2\pi f_{\mathrm{m}} t) \tag{4.44}$$

beschrieben werden. Dadurch, daß die Frequenz f_{m} nur kurzzeitig für die Dauer T_{h} eingenommen wird, ergibt sich das kontinuierliche Spektrum

$$\begin{aligned} S_{\mathrm{FH}}(f) &= A \cdot T_{\mathrm{h}} \cdot \mathrm{si}(\pi \cdot T_{\mathrm{h}} \cdot f) * \frac{-\mathrm{j}}{2} \cdot [\delta(f - f_m) - \delta(f + f_m] = \\ &= \frac{-\mathrm{j}A}{2} \cdot T_{\mathrm{h}} \cdot \mathrm{si}[\pi \cdot T_{\mathrm{h}} \cdot (f - f_{\mathrm{m}})] + \frac{\mathrm{j}A}{2} \cdot T_{\mathrm{h}} \cdot \mathrm{si}[\pi \cdot T_{\mathrm{h}} \cdot (f + f_{\mathrm{m}})] \end{aligned} \tag{4.45}$$

$S_{\mathrm{FH}}(f)$ liegt jeweils symmetrisch zu f_{m} und weist den gleichen Verlauf wie ein FSK-Spektrum nach Bild 4.30 auf. Damit es nicht zu einem Übersprechen kommt, muß auch bei FH - ähnlich wie bei FSK - ein Trägerabstand eingehalten werden, der dem Kehrwert der Verweildauer bzw. der 'Chipdauer' entspricht. In diesem Fall liegt ein orthogonaler Signalformsatz vor, und ein Matched-Filter-Empfänger ist in der Lage, eine perfekte Trennung benachbarter Spektren zu erzielen. Jede gewünschte Signalform aus einem Satz von N_{FH} Frequenzen wird 'optimal' empfangen, d.h. die zugehörige Autokorrelationsfunktion erreicht ein Maximum, während alle übrigen Null liefern. Wenn insgesamt N_{FH} Frequenzen im minimal zulässigen Abstand genutzt werden, dann wird von einem FH-System die Bandbreite

$$B_{\mathrm{FH}} = \frac{N_{\mathrm{FH}} + 1}{T_{\mathrm{h}}} = (N_{\mathrm{FH}} + 1) \cdot h_{\mathrm{r}} \tag{4.46}$$

belegt. Ein orthogonaler FH-Signalformsatz, der um eine in der Mitte des Übertragungsbandes der Bandbreite B_{FH} angesiedelte Frequenz f_0 gruppiert ist, wird z.B. bei ungeradem N_{FH} durch

$$s_{\mathrm{FHi}}(t) = A \cdot \mathrm{rect}\left(\frac{t}{T_{\mathrm{h}}}\right) \cdot \sin\left(2\pi\left(f_o + \left[i - \frac{N_{\mathrm{FH}} + 1}{2}\right] \cdot h_{\mathrm{r}}\right)t\right), \quad \text{mit} \quad i \in \{1 \ldots N_{\mathrm{FH}}\} \tag{4.47}$$

beschrieben. Die Signalform $s_{\mathrm{FH1}}(t)$ hat z.B. ein Spektrum symmetrisch zur tiefsten, am unteren Bandende angesiedelten Frequenz

$$f_0 - [(N_{\mathrm{FH}} - 1)/2] \cdot h_{\mathrm{r}} = f_{\mathrm{o}} - (B - h_{\mathrm{r}})/2.$$

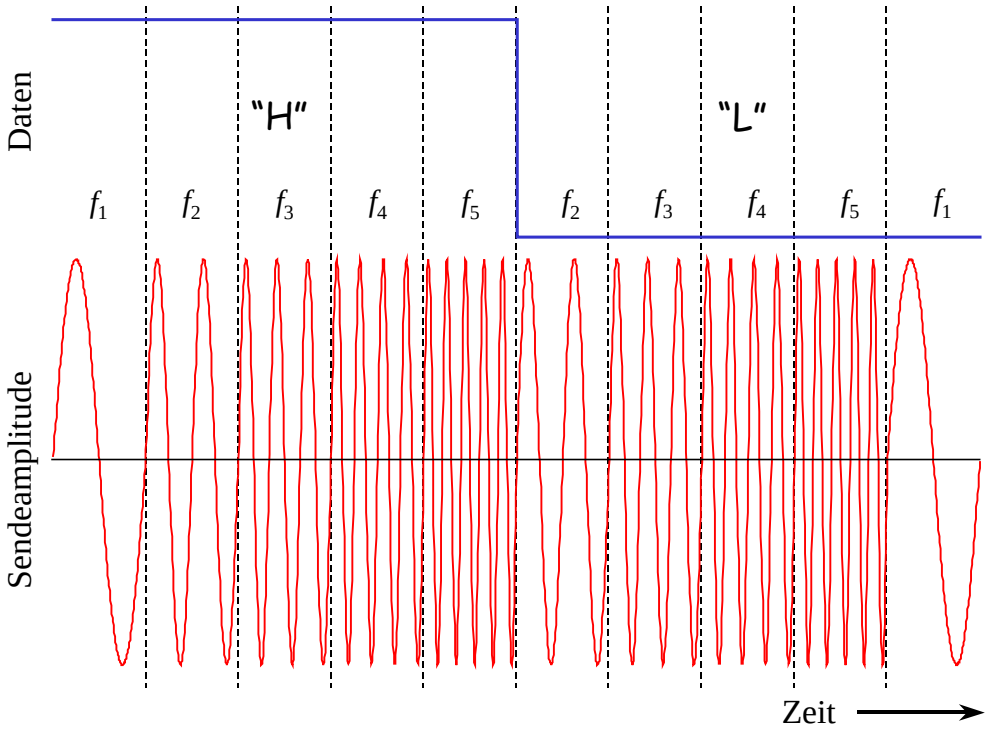
Am oberen Bandende liegt entsprechend mit $i = N_{\mathrm{FH}}$ die Signalform mit der Frequenz

$$f_0 + [(N_{\mathrm{FH}} - 1)/2] \cdot h_{\mathrm{r}} = f_0 + (B - h_{\mathrm{r}})/2.$$

FH kann als Erweiterung der FSK betrachtet werden. Das folgende Signal- und Systembeispiel – s. Bild 4.32 und Bild 4.33 soll die Zusammenhänge veranschaulichen. Dabei werden zur Übertragung binärer Daten nicht nur zwei, sondern fünf Frequenzen verwendet. Somit ist die Nutzinformation (d.h. ein Datenbit) an fünf separaten, diskreten Stellen im Übertragungsband präsent. Der Vorteil ist offensichtlich: Störungen bzw. Auslöschungen durch hohe Dämpfung an einer oder zwei dieser Stellen können der Datenübertragung nichts anhaben. Jedes gesendete Bit kann mit einer einfachen Mehrheitsentscheidung (drei aus fünf) im Empfänger fehlerfrei rekonstruiert werden.

Bild 4.32 illustriert das Beispiel, in dem ein "H"- und ein "L"-Datenbit und die jeweils zugeordneten Frequenzfolgen dargestellt sind. Das "H"-Bit wird dabei durch eine Folge aufsteigender Frequenzen f_1, f_2, f_3, f_4 und f_5 repräsentiert, die jeweils in einem festen Zeitschlitz (Chipdauer), der einem Fünftel der Bitdauer entspricht, gesendet werden. An den Chipgrenzen wechselt die Fre-

quenz sprungartig, jedoch ohne Einschwingvorgang oder Phasensprung. Aus Gründen einer übersichtlichen Darstellung sind die Frequenzwerte so gewählt, daß sich von Chip zu Chip die Zahl der Schwingungen um Eins erhöht. Für das “L”-Bit werden die gleichen Frequenzen, jedoch in anderer zeitlicher Reihenfolge verwendet, d.h. f_2, f_3, f_4, f_5 und f_1.



Die Frequenzzuordnung nach Bild 4.32 ist nicht die einzige Möglichkeit, denn mit den fünf Frequenzen als Binärvariablen lassen sich 2^5=32 verschiedene Kombinationen zusammenstellen, von denen nur zwei genutzt werden. Diese hohe Redundanz ist letztlich für die Störresistenz verantwortlich.

Bild 4.32: Signalbeispiel für FH mit 5 Frequenzen pro Bit

Bei der Wahl der beiden Kombinationen für “L”- und “H”-Bit muß man zur Erzielung bestmöglicher Störresistenz beachten, daß die Frequenzen in den jeweiligen Zeitschlitzen immer verschieden sind. Auch unter dieser Randbedingung gibt es mehr als nur zwei Kombinationen, so daß man mit den fünf Frequenzen ohne Einbuße an Störresistenz mehr als ein Bit parallel übertragen könnte.

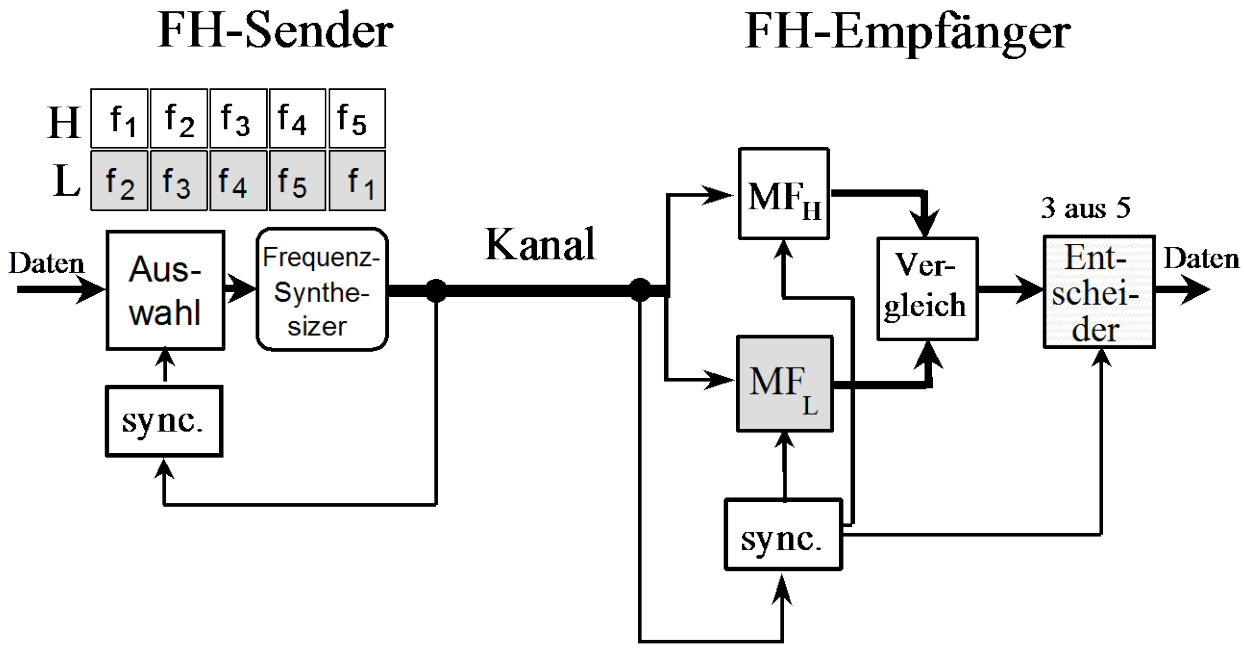


Bild 4.33: FH-Systembeispiel

5 Repetitorium zur Fouriertransformation

5.1 Die Fourierreihe

In der technischen Literatur wird die Fourierreihe normalerweise unabhängig vom Fourierintegral behandelt. Es ist für das Verständnis der Zusammenhänge jedoch vorteilhafter, die Fourierreihe als Spezialfall des Fourierintegrals darzustellen. Für das Verstehen der diskreten Fouriertransformation ist darüber hinaus die Behandlung von Abtastsignalen von Bedeutung. Im folgenden wird die Beziehung der Fourierreihe zum Fourierintegral sowie die bereits im Zusammenhang mit der A/D-Wandlung betrachtete Fouriertransformation von Abtastsignalen vertiefend diskutiert. Damit wird der notwendige Rahmen für die Herleitung der diskreten Fouriertransformation (DFT) und deren schnelle Berechnungsmethode, die FFT (Fast Fourier Transform) vorbereitet.

$h(t)$ sei eine periodische Zeitfunktion, wobei $f_0=1/T_0$ die Grundfrequenz von $h(t)$ ist. Dann läßt sich $h(t)$ in Form der bekannten Fourierreihe

$$h(t)=\frac{a_o}{2}+\sum_{n=1}^{\infty}\left[a_{\mathrm{n}}\cos\left(2\pi nf_0t\right)+b_{\mathrm{n}}\sin\left(2\pi nf_0t\right)\right] \tag{5.1}$$

darstellen. Die Amplituden der Sinus- und Cosinusfunktionen die mit den Koeffizienten der Fourierreihe identisch sind, sind gegeben durch die Integrale

$$a_{\mathrm{n}}=\frac{2}{T_0}\cdot\int_{-T_0/2}^{T_0/2}h(t)\cos\left(2\pi nf_0t\right)dt,\quad n=0,1,2,... \tag{5.2}$$

$$b_{\mathrm{n}}=\frac{2}{T_0}\cdot\int_{-T_0/2}^{T_0/2}h(t)\sin\left(2\pi nf_0t\right)dt,\quad n=0,1,2,... \tag{5.3}$$

Unter Anwendung der Identitäten

$$\cos\left(2\pi nf_0t\right)=\frac{1}{2}\left(\mathrm{e}^{\mathrm{j}2\pi nf_0t}+\mathrm{e}^{-\mathrm{j}2\pi nf_0t}\right)\qquad \sin\left(2\pi nf_0t\right)=\frac{1}{2\mathrm{j}}\left(\mathrm{e}^{\mathrm{j}2\pi nf_0t}-\mathrm{e}^{-\mathrm{j}2\pi nf_0t}\right)$$

läßt sich (5.1) umschreiben in:

$$h(t)=\frac{a_0}{2}+\frac{1}{2}\sum_{n=1}^{\infty}\left(a_{\mathrm{n}}-\mathrm{j}b_{\mathrm{n}}\right)\cdot\mathrm{e}^{\mathrm{j}2\pi nf_0t}+\frac{1}{2}\sum_{n=1}^{\infty}\left(a_{\mathrm{n}}+\mathrm{j}b_{\mathrm{n}}\right)\cdot\mathrm{e}^{-\mathrm{j}2\pi nf_0t} \tag{5.4}$$

Zur Vereinfachung dieses Ausdrucks werden in (5.2) und (5.3) negative Werte für n eingeführt.

$$a_{-\mathrm{n}}=\frac{2}{T_0}\int_{-\frac{T_0}{2}}^{\frac{T_0}{2}}h(t)\cdot\cos\left(-2\pi nf_0t\right)dt=\frac{2}{T_0}\int_{-\frac{T_0}{2}}^{\frac{T_0}{2}}h(t)\cdot\cos\left(2\pi nf_0t\right)dt=a_{\mathrm{n}},\quad n=1,2,3,... \tag{5.5}$$

$$b_{-\mathrm{n}}=\frac{2}{T_0}\int_{-\frac{T_0}{2}}^{\frac{T_0}{2}}h(t)\cdot\sin\left(-2\pi nf_0t\right)dt=-\frac{2}{T_o}\int_{-\frac{T_0}{2}}^{\frac{T_0}{2}}h(t)\cdot\sin\left(2\pi nf_0t\right)dt=-b_{\mathrm{n}},\quad n=1,2,3,... \tag{5.6}$$

Damit erhält man aus (5.4)

$$\sum_{n=1}^{\infty}a_{\mathrm{n}}\cdot\mathrm{e}^{-\mathrm{j}2\pi nf_0t}=\sum_{n=-1}^{-\infty}a_{\mathrm{n}}\cdot\mathrm{e}^{\mathrm{j}2\pi nf_0t} \tag{5.7}$$

und

$$\sum_{n=1}^{\infty} \mathrm{j}b_{\mathrm{n}} \cdot \mathrm{e}^{-\mathrm{j}2\pi n f_0 t} = -\sum_{n=-1}^{-\infty} \mathrm{j}b_{\mathrm{n}} \cdot \mathrm{e}^{\mathrm{j}2\pi n f_0 t} \,. \tag{5.8}$$

Das Einsetzen von (5.7) und (5.8) in (5.4) ergibt dann

$$h(t) = \frac{a_0}{2} + \frac{1}{2} \cdot \sum_{\substack{n=-\infty \\ n\neq 0}}^{\infty} (a_{\mathrm{n}} - \mathrm{j}b_{\mathrm{n}}) \cdot \mathrm{e}^{\mathrm{j}2\pi n f_0 t} = \sum_{n=-\infty}^{\infty} \alpha_{\mathrm{n}} \cdot \mathrm{e}^{\mathrm{j}2\pi n f_0 t} \tag{5.9}$$

Dies ist die exponentielle Darstellungsform der Fourierreihe. Die Koeffizienten α_{n} sind im allgemeinen komplex. Mit Einsetzen von (5.2) und (5.3) erhält man

$$\alpha_n = \frac{1}{2}(a_{\mathrm{n}} - \mathrm{j}b_{\mathrm{n}}) = \frac{1}{T_0} \cdot \left[\int_{-T_0/2}^{T_0/2} h(t) \cos(2\pi n f_0 t)\, dt - \mathrm{j} \int_{-T_0/2}^{T_0/2} h(t) \sin(2\pi n f_0 t)\, dt \right]. \tag{5.10}$$

Das Ersetzen von cos(·) und sin(·) unter Verwendung der Identitäten

$$\cos(2\pi n f_0 t) = \frac{1}{2}\left(\mathrm{e}^{\mathrm{j}2\pi n f_0 t} + \mathrm{e}^{-\mathrm{j}2\pi n f_0 t}\right) \qquad \sin(2\pi n f_0 t) = \frac{1}{2\mathrm{j}}\left(\mathrm{e}^{\mathrm{j}2\pi n f_0 t} - \mathrm{e}^{-\mathrm{j}2\pi n f_0 t}\right)$$

liefert das Zwischenergebnis

$$\alpha_n = \frac{1}{2T_0} \cdot \left[\int_{-T_0/2}^{T_0/2} h(t) \cdot \mathrm{e}^{\mathrm{j}2\pi f_0 t} dt + \int_{-T_0/2}^{T_0/2} h(t) \cdot \mathrm{e}^{-\mathrm{j}2\pi f_0 t} dt - \int_{-T_0/2}^{T_0/2} h(t) \cdot \mathrm{e}^{\mathrm{j}2\pi f_0 t} dt + \int_{-T_0/2}^{T_0/2} h(t) \cdot \mathrm{e}^{-\mathrm{j}2\pi f_0 t} dt \right]$$

aus dem sofort

$$\alpha_{\mathrm{n}} = \frac{1}{T_0} \int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} h(t) \cdot \mathrm{e}^{-\mathrm{j}2\pi n f_0 t} dt \quad \text{für} \quad n=0,\pm 1,\pm 2,... \tag{5.11}$$

folgt. Damit kann die exponentielle Darstellungsform der Fourierreihe auf Basis der komplexen Koeffizienten α_{n} angegeben werden:

$$h(t) = \frac{a_0}{2} + \frac{1}{2} \cdot \sum_{\substack{n=-\infty \\ n\neq 0}}^{\infty} (a_{\mathrm{n}} - \mathrm{j}b_{\mathrm{n}}) \cdot \mathrm{e}^{\mathrm{j}2\pi n f_0 t} = \sum_{n=-\infty}^{\infty} \alpha_{\mathrm{n}} \cdot \mathrm{e}^{\mathrm{j}2\pi n f_0 t} \tag{5.12}$$

5.2 Das Fourierintegral und die diskrete FT

Das Fourierintegral ermöglicht – sofern es konvergiert - die Berechnung der Transformierten beliebiger, insbesondere auch nichtperiodischer Zeitsignale. Für ein Zeitsignal $h(t)$ ist es definiert durch

$$H(f) = \int_{-\infty}^{\infty} h(t) \cdot \mathrm{e}^{-\mathrm{j}2\pi f t} dt \tag{5.13}$$

Formal hat (5.13) große Ähnlichkeit mit (5.11). Der wesentliche Unterschied ist jedoch, daß sich die Integration bei (5.11) exakt über eine Periodendauer des Zeitsignals $h(t)$ erstreckt, während in (5.13) von $-\infty$ ... $+\infty$ zu integrieren ist. Das bedeutet, daß bei (5.11) die gesamte Information über das Zeitsignal in einer Periode enthalten sein muß, was naturgemäß nur bei periodischen Signalen

geben ist. Mit dieser – für die Praxis erheblichen – Einschränkung wird die Fourierreihe normalerweise in Lehrbüchern und Lehrveranstaltungen behandelt. Wir werden sehen, daß die Eigenschaft der Periodizität eines Zeitsignals eine ganz essentielle Grundlage für die korrekte Ausführung der diskreten Fouriertransformation (DFT) ist. Deshalb ist es für das Verständnis der DFT und ihre fehlerfreie Anwendung sehr hilfreich, den Zusammenhang mit der Fourierreihe detailliert zu betrachten. Bevor dies in einer kombinierten theoretischen und anschaulichen Analyse vorgenommen wird, sind zunächst einige wichtige Eigenschaften der kontinuierlichen FT zusammengestellt und werden ihren diskreten Entsprechungen gegenübergestellt. Danach wird der in der Praxis am häufigsten verwendete Algorithmus zur Ausführung der schnellen Fouriertransformation – der Cooley-Tukey-Algorithmus zur Basis 2 – grundlegend untersucht.

Zum Abschluß dieses Abschnitts werden die Zusammenhänge zwischen der DFT und der Fourierreihe sowohl mathematisch als auch anschaulich untersucht und dargestellt.

Fouriertransformation:

Im folgenden wird zur Darstellung der diskreten Zeit die Variable k und für die Frequenz die Variable n verwendet. Beim praktischen Umgang – d.h. für konkrete Berechnungen – ist dabei zu beachten, daß weder die diskrete Zeit noch die Frequenz dimensionslos sind. Durch die Signalabtastung ist stets ein Zeitraster $T_A=1/f_A$ vorgegeben, so daß $k := k \cdot T_A$ gilt. Für die Frequenz ist das Raster – wie weiter unten noch näher erklärt wird – $n := n/N \cdot T_A$, d.h. die Frequenzauflösung wird durch den Kehrwert der Gesamtdauer des zu transformierenden Zeitsignalausschnitts bestimmt.

$$\underbrace{H(f)=\int_{-\infty}^{\infty} h(t)\cdot e^{-j2\pi f t}dt}_{\text{kontinuierlich}} \qquad \underbrace{H(n)=\sum_{k=0}^{N-1} h(k)\cdot e^{-j2\pi nk/N}\ ,\ n=0,1,...,N-1}_{\text{diskret}} \tag{5.14}$$

Inverse Fouriertransformation:

$$\underbrace{h(t)=\int_{-\infty}^{\infty} H(f)\cdot e^{j2\pi f t}df}_{\text{kontinuierlich}} \qquad \underbrace{h(k)=\frac{1}{N}\sum_{n=0}^{N-1} H(n)\cdot e^{j2\pi nk/N}\ ,\ k=0,1,...,N-1}_{\text{diskret}} \tag{5.15}$$

5.2.1 Eigenschaften der kontinuierlichen und diskreten Fouriertransformation

Die Eigenschaften der kontinuierlichen Fouriertransformation lassen sich auf die diskrete Fouriertransformation (DFT) übertragen. Dies beruht auf der Tatsache, daß die DFT ein Spezialfall der kontinuierlichen Fouriertransformation ist. Es wäre daher möglich, die hergeleiteten Eigenschaften einfach mit einer für die DFT geeigneten Ausdrucksweise neu zu formulieren. Andererseits ist aber etwas Praxis im Umgang mit zeitdiskreten Funktionen nützlich, vor allem in Hinblick auf die korrekte Skalierung. Aus diesem Grund werden im folgenden einige wichtige Eigenschaften DFT von einem zeitdiskreten Ansatz her entwickelt.

Zur Vereinfachung der Schreibweise wird im weiteren – wie schon oben eingeführt – kT_A durch k und n/NT_A durch n ersetzt.

Linearität:

Kontinuierliche FT: $x(t)+y(t) \quad \circ\!\!-\!\!\bullet \quad X(f)+Y(f)$ (5.16)

Wenn $X(n)$ und $Y(n)$ die diskreten Fouriertransformierten von $x(k)$ und $y(k)$ sind, dann gilt

$$x(k)+y(k) \circ\!\!-\!\!\bullet\, X(n)+Y(n) \tag{5.17}$$

Symmetrie:

Wenn $h(t)$ und $H(f)$ ein FT-Paar bilden und entsprechend $h(k)$ und $H(n)$ ein DFT-Paar sind, dann gelten die Beziehungen

$$H(t) \;\circ\!\!-\!\!\bullet\; h(-f) \qquad \frac{1}{N}H(k) \;\circ\!\!-\!\!\bullet\; h(-n) \tag{5.18}$$

Der Beweis für die DFT folgt z.B. direkt durch Einsetzen von $-k$ statt k in (5.15)

$$h(-k)=\frac{1}{N}\sum_{k=0}^{N-1}H(n)\cdot \mathrm{e}^{\mathrm{j}2\pi n(-k)/N} \tag{5.19}$$

und Austauschen von k und n:

$$h(-n)=\frac{1}{N}\sum_{k=0}^{N-1}H(k)\cdot \mathrm{e}^{-\mathrm{j}2\pi nk/N} \tag{5.20}$$

Zeitverschiebung:

Bei einer kontinuierlichen Zeitverschiebung t_0 von $h(t)$, bzw. einer diskreten Verschiebung i von $h(k)$ erhält man die Beziehungen

$$h(t-t_0) \circ\!\!-\!\!\bullet\, H(f)\cdot \mathrm{e}^{-\mathrm{j}2\pi f t_0} \qquad h(k-i) \circ\!\!-\!\!\bullet\, H(n)\cdot \mathrm{e}^{-\mathrm{j}2\pi ni/N} \tag{5.21}$$

Eine Zeitverschiebung hat eine Änderung der Phase zu Folge.

Den Beweis bei der DFT liefert z.B. die Substitution $r=k-i$ in (5.15):

$$\begin{aligned} h(r)&=\frac{1}{N}\sum_{n=0}^{N-1}H(n)\cdot \mathrm{e}^{\mathrm{j}2\pi nr/N} \quad \Rightarrow \\ h(k-i)&=\frac{1}{N}\sum_{n=0}^{N-1}H(n)\cdot \mathrm{e}^{\mathrm{j}2\pi n(k-i)/N}=\frac{1}{N}\sum_{n=0}^{N-1}\left\{H(n)\cdot \mathrm{e}^{-\mathrm{j}2\pi ni/N}\right\}\cdot \mathrm{e}^{\mathrm{j}2\pi nk/N} \end{aligned} \tag{5.22}$$

Frequenzverschiebung (Modulationstheorem):

Wenn man die kontinuierliche Übertragungsfunktion $H(f)$ um die Frequenz f_0 verschiebt, bzw. die diskrete Übertragungsfunktion $H(n)$ einer diskreten Verschiebung i unterzieht, erhält man die Beziehungen

$$h(t)\cdot \mathrm{e}^{\mathrm{j}2\pi f_0 t} \circ\!\!-\!\!\bullet\, H(f-f_0) \qquad h(k)\cdot \mathrm{e}^{\mathrm{j}2\pi ik/N} \circ\!\!-\!\!\bullet\, H(n-i) \tag{5.23}$$

Den Beweis, wiederum für die DFT, liefert die Substitution $r=n-i$ in (5.14):

$$\begin{aligned} H(r)&=\sum_{k=0}^{N-1}h(k)\cdot \mathrm{e}^{-\mathrm{j}2\pi rk/N} \quad \Rightarrow \\ H(n-i)&=\sum_{k=0}^{N-1}h(k)\cdot \mathrm{e}^{-\mathrm{j}2\pi (n-i)k/N}=\sum_{k=0}^{N-1}\left\{h(k)\cdot \mathrm{e}^{\mathrm{j}2\pi ik/N}\right\}\cdot \mathrm{e}^{-\mathrm{j}2\pi nk/N} \end{aligned} \tag{5.24}$$

Im weiteren wird auf die Gegenüberstellung der korrespondierenden Beziehungen für kontinuierliche FT und die DFT verzichtet und wegen ihrer Bedeutung für die digitale Signalverarbeitung nur noch die diskrete Darstellung betrachtet.

Alternative Inversionsbeziehung:

Die Gleichung für die inverse diskreten Fouriertransformation nach (5.15) läßt sich in folgende Form umschreiben:

$$h(k) = \frac{1}{N}\left[\sum_{k=0}^{N-1} H^*(n)\cdot \mathrm{e}^{-\mathrm{j}2\pi nk/N}\right]^*, \tag{5.25}$$

wobei $*$ bedeutet, daß der entsprechende Ausdruck konjugiert-komplex zu nehmen ist. Zum Beweis wird $H(n)=R(n)+\mathrm{j}\,I(n)$ gesetzt, woraus $H^*(n)=R(n)-\mathrm{j}\,I(n)$ folgt. Durch Einsetzen in (5.25) erhält man

$$\begin{aligned}
h(k) &= \frac{1}{N}\left[\sum_{n=0}^{N-1}\left[R(n)-\mathrm{j}\cdot I(n)\right]\mathrm{e}^{-\mathrm{j}2\pi nk/N}\right]^* \\
&= \frac{1}{N}\left\{\sum_{n=0}^{N-1}\left[R(n)-\mathrm{j}\cdot I(n)\right]\cdot\left[\cos\frac{2\pi nk}{N}-\mathrm{j}\cdot\sin\frac{2\pi nk}{N}\right]\right\}^* \\
&= \frac{1}{N}\left\{\sum_{n=0}^{N-1}\left(R(n)\cdot\cos\frac{2\pi nk}{N}-I(n)\cdot\sin\frac{2\pi nk}{N}\right)-\mathrm{j}\cdot\sum_{n=0}^{N-1}\left(R(n)\cdot\sin\frac{2\pi nk}{N}+I(n)\cdot\cos\frac{2\pi nk}{N}\right)\right\}^* \\
&= \frac{1}{N}\left\{\sum_{n=0}^{N-1}\left(R(n)\cdot\cos\frac{2\pi nk}{N}-I(n)\cdot\sin\frac{2\pi nk}{N}\right)+\mathrm{j}\sum_{n=0}^{N-1}\left(R(n)\cdot\sin\frac{2\pi nk}{N}+I(n)\cdot\cos\frac{2\pi nk}{N}\right)\right\} \\
&= \frac{1}{N}\sum_{n=0}^{N-1}\left[\underbrace{R(n)+\mathrm{j}\cdot I(n)}_{H(n)}\right]\cdot\left(\cos\frac{2\pi nk}{N}+\mathrm{j}\cdot\sin\frac{2\pi nk}{N}\right)=\frac{1}{N}\sum_{n=0}^{N-1}H(n)\cdot\mathrm{e}^{\mathrm{j}2\pi nk/N}
\end{aligned} \tag{5.26}$$

Der Vorteil dieser alternativen Inversionsbeziehung liegt darin, daß die Darstellung nach (5.25) sich sowohl zur Berechnung der DFT als auch der IDFT Beziehung verwenden läßt. Wenn die Berechnung auf einem Digitalrechner erfolgt, braucht man nur ein Programm zu erstellen, das mit geringer Modifikation beide Ergebnisse liefern kann. Es sind lediglich Ausdrücke konjugiert-komplex zu nehmen, was in einfacher Weise durch Umkehren des Vorzeichens beim Imaginärteil geschieht.

Faltungstheorem für den Frequenzbereich

Für die diskrete Faltung im Frequenzbereich gilt der folgende Ansatz

$$Y(n) = \sum_{i=0}^{N-1} X(i)\cdot H(n-i) \tag{5.27}$$

Der Beweis erfolgt, indem zunächst die DFTs zur Gewinnung von $X(i)$ und $H(n-i)$ substituiert werden:

$$Y(n)=\sum_{i=0}^{N-1}X(i)\cdot H(n-i)=\sum_{i=0}^{N-1}\left[\sum_{m=0}^{N-1}x(m)\cdot \mathrm{e}^{-\mathrm{j}2\pi mi/N}\right]\cdot\left[\sum_{k=0}^{N-1}h(k)\cdot \mathrm{e}^{-\mathrm{j}2\pi k(n-i)/N}\right]$$

Das Zusammenfassen der von *i* abhängigen Terme unter der Summe über *i* liefert:

$$Y(n)=\sum_{m=0}^{N-1}\sum_{k=0}^{N-1}x(m)\cdot h(k)\cdot \mathrm{e}^{-\mathrm{j}2\pi kn/N}\left[\sum_{i=0}^{N-1}\mathrm{e}^{-\mathrm{j}2\pi mi/N}\cdot \mathrm{e}^{\mathrm{j}2\pi ki/N}\right] \tag{5.28}$$

Der Term in eckigen Klammern entspricht der ***Orthogonalitätsbeziehung,*** die besagt, daß eine Summe über eine ganze Anzahl von Perioden eines Produkts sinusförmiger Signale verschwindet, wenn die Frequenzen verschieden sind. Für den Fall nach (5.28) bedeutet dies, daß bei den Exponentialfunktionen Orthogonalität vorliegt, wenn *m* und *k* verschieden sind. D.h. nur für den Fall *m=k* ergibt sich ein von Null verschiedener Wert. Für *m=k* wird das Produkt gleich Eins, so daß die Summe *N* ergibt. Damit folgt aus (5.28)

$$Y(n)=\sum_{i=0}^{N-1}X(i)\cdot H(n-i)=N\sum_{k=0}^{N-1}x(k)\cdot h(k)\cdot \mathrm{e}^{-\mathrm{j}2\pi nk/N} \tag{5.29}$$

Man hat also diskrete Fouriertransformationspaar

$$x(k)\cdot h(k)\;\circ\!\!-\!\!\bullet\;\frac{1}{N}\sum_{i=0}^{N-1}X(i)\cdot H(n-i)\,. \tag{5.30}$$

<u>Das Theorem der diskreten Korrelation:</u>

Das Transformationspaar

$$\sum_{i=0}^{N-1}x(i)\cdot h(k+i)\quad\circ\!\!-\!\!\bullet\quad X^{*}(n)\cdot H(n) \tag{5.31}$$

wird diskretes Korrelationstheorem genannt. Damit läßt sich eine Zeitbereichs-Korrelation in äquivalenter Weise auch im Frequenzbereich ausführen. Zum Beweis setzt man die Beziehung der IDFT für *x*(*i*) und *h*(*k*+*i*) im linken Ausdruck von (5.31) an und erhält

$$\sum_{i=0}^{N-1}\underbrace{\left[\frac{1}{N}\sum_{n=0}^{N-1}X(n)\cdot \mathrm{e}^{\mathrm{j}2\pi in/N}\right]}_{x(i)}\cdot\underbrace{\left[\frac{1}{N}\sum_{m=0}^{N-1}H(m)\cdot \mathrm{e}^{\mathrm{j}2\pi m(k+i)/N}\right]}_{h(k+i)} \tag{5.32}$$

Um *X**(*n*) einzuführen, wird der Inhalt der linken eckigen Klammer konjugiert-komplex genommen und das Ganze außen an der Klammer wieder aufgehoben:

$$\sum_{i=0}^{N-1}x(i)\cdot h(k+i)=\sum_{i=0}^{N-1}\underbrace{\left[\frac{1}{N}\sum_{n=0}^{N-1}X^{*}(n)\cdot \mathrm{e}^{-\mathrm{j}2\pi in/N}\right]^{*}}_{x(i)}\cdot\left[\frac{1}{N}\sum_{m=0}^{N-1}H(m)\cdot \mathrm{e}^{\mathrm{j}2\pi m(k+i)/N}\right] \tag{5.33}$$

Für den Fall, daß nur reelle Zeitfunktionen *x*(*i*) betrachtet werden, kann die Bildung des konjugiert-komplexen Klammerinhaltes entfallen. Dann kann (5.33) unter Zusammenfassung der nur von *i* abhängigen Terme unter der entsprechenden Summe umgeschrieben werden:

$$\sum_{i=0}^{N-1}x(i)\cdot h(k+i)=\frac{1}{N}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1}X^{*}(n)\cdot H(m)\cdot \mathrm{e}^{\mathrm{j}2\pi mk/N}\cdot\left[\frac{1}{N}\sum_{i=0}^{N-1}\mathrm{e}^{-\mathrm{j}2\pi in/N}\cdot \mathrm{e}^{\mathrm{j}2\pi im/N}\right] \tag{5.34}$$

In der linken eckigen Klammer findet sich die bekannte ***Orthogonalitätsbeziehung*** wieder, so daß das Produkt der Exponentialfunktionen stets verschwindet, außer für $n = m$. Dann wird das Produkt gleich Eins, so daß die Summe wiederum N ergibt. Damit folgt aus (5.34)

$$\sum_{i=0}^{N-1} x(i)\cdot h(k+i) = \frac{1}{N}\sum_{n=0}^{N-1} \underbrace{X^*(n)\cdot H(n)}_{\bullet\!\!-\!\!\circ\ \sum_{i=0}^{N-1} x(i)\cdot h(k+i)} \cdot \mathrm{e}^{\,\mathrm{j}2\pi nk/N} \tag{5.35}$$

PARSEVALsches Theorem:

Für diskrete Funktionen ist der Zusammenhang zwischen der Energie im Zeit- und im Frequenzbereich gegeben durch

$$\sum_{k=0}^{N-1} h^2(k) = \frac{1}{N}\sum_{n=0}^{N-1} \left|H(n)\right|^2 \tag{5.36}$$

Zum Beweis setze man in (5.35) $x(i)=h(i)$ und $k=0$, dann hat man bereits das gesuchte Ergebnis:

$$\sum_{i=0}^{N-1} h(i)\cdot h(i) = \sum_{i=0}^{N-1} h^2(i) = \frac{1}{N}\sum_{n=0}^{N-1} H^*(n)\cdot H(n)\cdot \mathrm{e}^0 = \frac{1}{N}\sum_{n=0}^{N-1} \left|H(n)\right|^2 \tag{5.37}$$

5.3 Herleitung des Basis 2-FFT-Algorithmus (Cooley-Tukey)

In diesem Abschnitt wird auf der Grundlage theoretischer Überlegungen der FFT-Algorithmus für $N=2^\gamma$ hergeleitet. Zum Einstieg wird zunächst der Fall $\gamma=2$, also $N=4$ untersucht. Dann erfolgt die allgemeine Erweiterung mit der Herleitung des Algorithmus für den Fall $N=2^\gamma$.

Erklärung der verwendeten Begriffe

Häufig erfordert eine theoretische Darlegung die Einführung neuer und ungewohnter Ausdrucksweisen. Im Falle der FFT ist die Vereinfachung, die man dadurch erreichen kann, in jedem Fall der Mühe wert. Man betrachte die bekannte Beziehung der diskreten Fouriertransformation in etwa abgekürzter Schreibweise

$$X(n) = \sum_{k=0}^{N-1} x_0(k)\cdot W^{nk}, \quad n = 0,1,...N-1, \tag{5.38}$$

wobei $W=\mathrm{e}^{-\mathrm{j}2\pi/N}$ ist. W enthält also alle Konstanten und wird auch Drehfaktor[10] genannt, ein Begriff, der im weiteren noch verdeutlicht wird. Für die folgenden Betrachtungen stellt es sich als Schlüssel heraus, die natürlichen Zahlen n und k in ***Binärschreibweise*** darzustellen. Wenn also $N=4$ ist, dann ist $\gamma=2$ und k und n sind als 2-bit-Binärzahlen wie folgt darstellbar:

k=0, 1, 2, 3	oder	$k=(k_1,k_0)$ = 00, 01, 10, 11
n=0, 1, 2, 3	oder	$n=(n_1,n_0)$ = 00, 01, 10, 11

[10] engl.: Twiddle Factor

Eine formale Kurzschreibweise für *k* und *n* ist

$$k = 2 \cdot k_1 + k_0, \qquad n = 2 \cdot n_1 + n_0 \tag{5.39}$$

Unter Verwendung von (5.39) kann (5.38) für den Fall *N*=4 wie folgt umgeschrieben werden:

$$X\left(n_1, n_0\right) = \sum_{k_0=0}^{1} \sum_{k_1=0}^{1} x_0\left(k_1, k_0\right) \cdot W^{(2n_1+n_0)\cdot(2k_1+k_0)} \tag{5.40}$$

Man beachte, daß die eine Summation aus (5.38) nun zur Berücksichtigung aller Bits von *k* durch γ Summationen zu ersetzen ist.

Faktorisierung von W^P

Nun wird man der Term $W^{(2n_1+n_0)\cdot(2k_1+k_0)}$ näher betrachtet. Wegen $W^{a+b} = W^a \cdot W^b$ erhält man

$$W^{(2n_1+n_0)2k_1} \cdot W^{(2n_1+n_0)k_0} = \left(W^{4n_1k_1}\right) \cdot W^{2n_0 \cdot k_1} \cdot W^{(2n_1+n_0)k_0} = W^{2n_0k_1} \cdot W^{(2n_1+n_0)k_0} \tag{5.41}$$

Man beachte, daß der erste Term in Klammern gleich Eins ist, da

$$W^{4n_1k_1} = \left(W^4\right)^{n_1k_1} = \left[\mathrm{e}^{-\mathrm{j}2\pi 4/4}\right]^{n_1k_1} = (1)^{n_1k_1} = 1 \tag{5.42}$$

gilt. Somit kann (5.38) in der Form

$$X\left(n_1, n_0\right) = \sum_{k_0=0}^{1} \left[\sum_{k_1=0}^{1} x_0\left(k_1, k_0\right) \cdot W^{2n_0k_1} \right] \cdot W^{(2n_1+n_0)k_0} \tag{5.43}$$

geschrieben werden Diese Gleichung bildet das Fundament des FFT-Algorithmus. Um sich dies weiter zu verdeutlichen, betrachte man die beiden Teilsummen aus (5.43) einzeln. Zunächst die innere Summe in den eckigen Klammern:

$$x_1\left(n_0, k_0\right) = \sum_{k_1=0}^{1} x_0\left(k_1, k_0\right) \cdot W^{2n_0 \cdot k_1} \tag{5.44}$$

Durch Einsetzen der Zahlen ergeben sich folgende vier Gleichungen:

$$x_1(0,0) = x_0(0,0) + x_0(1,0)W^0$$
$$x_1(0,1) = x_0(0,1) + x_0(1,1)W^0$$
$$x_1(1,0) = x_0(0,0) + x_0(1,0)W^2$$
$$x_1(1,1) = x_0(0,1) + x_0(1,1)W^2$$

Es ist naheliegend (5.44) in Matrixform darzustellen.

$$\begin{bmatrix} x_1(0,0) \\ x_1(0,1) \\ x_1(1,0) \\ x_1(1,1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \cdot \begin{bmatrix} x_0(0,0) \\ x_0(0,1) \\ x_0(1,0) \\ x_0(1,1) \end{bmatrix} \tag{5.45}$$

Wird jetzt in einem nächsten Schritt die äußere Summe von (5.43) in die Form

$$x_2(n_0,n_1) = \sum_{k_0=0}^{1} x_1(n_0,k_0) \cdot W^{(2n_1+n_0)k_0} \tag{5.46}$$

umgeschrieben und werden die Zahlen eingesetzt, dann erhält man in ähnlicher Weise wie oben ein Gleichungssystem, das sich wieder in Matrixform darstellen läßt.

$$\begin{bmatrix} x_2(0,0) \\ x_2(0,1) \\ x_2(1,0) \\ x_2(1,1) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} x_1(0,0) \\ x_1(0,1) \\ x_1(1,0) \\ x_1(1,1) \end{bmatrix} \tag{5.47}$$

Aus (5.43) und (5.46) folgt schließlich

$$X(n_1,n_0) = x_2(n_0,n_1). \tag{5.48}$$

Wie man sieht treten die Endergebnisse $x_2(n_0,n_1)$, die man aus der äußeren Summation erhält, verglichen mit den gewünschten Größen $X(n_1,n_0)$, in ***Bit-Umkehr-Reihenfolge*** auf. Dies erfordert eine Umordnung im Ergebnis (Bit Reversing), die dem FFT-Algorithmus eigen ist.

Mit der Zusammenstellung der Gleichungen (5.44) (5.46) und (5.48)

$$\begin{aligned} x_1(n_0,k_0) &= \sum_{k_1=0}^{1} x_0(k_1,k_0) \cdot W^{2n_0k_1} \\ x_2(n_0,n_1) &= \sum_{k_0=0}^{1} x_1(n_0,k_0) \cdot W^{(2n_1+n_0)k_0} \\ X(n_1,n_0) &= x_2(n_0,n_1) \end{aligned} \tag{5.49}$$

bildet (5.49) die ursprüngliche Form des ***Cooley-Tukey-FFT-Algorithmus*** für N=4. Das Gleichungssystem wird sukzessiv berechnet, d.h. die Auswertung einer Folgegleichung benötigt die Ergebnisse aus der vorangehenden Gleichung.

Beispiel:

Um sich mit der beim ***Cooley-Tukey-FFT-Algorithmus*** verwendeten Darstellungsweise weiter vertraut zu machen, wird ein Beispiel mit einer FFT der Länge $N=2^3=8$ detailliert untersucht. Entsprechend (5.39) hat man jetzt

$$\begin{aligned} n &= 4n_2 + 2n_1 + n_0, \ \text{mit} \quad n_i \in \{0,1\} \\ k &= 4k_2 + 2k_1 + k_0, \ \text{mit} \quad k_i \in \{0,1\} \end{aligned}, \tag{5.50}$$

und damit liefert (5.38)

$$X\left(n_2, n_1, n_0\right) = \sum_{k_0=0}^{1} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} x_0\left(k_2, k_1, k_0\right) \cdot W^{\left(4n_2+2n_1+n_0\right)\cdot\left(4k_2+2k_1+k_0\right)}. \tag{5.51}$$

Jetzt wird der Exponent des Drehfaktors *W* ausgeschrieben:

$$W^{\left(4n_2+2n_1+n_0\right)\cdot\left(4k_2+2k_1+k_0\right)} = W^{\left(4n_2+2n_1+n_0\right)\cdot 4k_2} \cdot W^{\left(4n_2+2n_1+n_0\right)\cdot 2k_1} \cdot W^{\left(4n_2+2n_1+n_0\right)\cdot k_0} \tag{5.52}$$

Wegen $W^8 = \left(\mathrm{e}^{\mathrm{j}2\pi/8}\right)^8 = 1$ gilt

$$\begin{aligned} W^{\left(4n_2+2n_1+n_0\right)\cdot 4k_2} &= \left[W^{8\cdot\left(2n_2k_2\right)}\right] \cdot \left[W^{8\cdot\left(n_1k_2\right)}\right] W^{4n_0k_2} = W^{4n_0k_2} \\ &\text{und} \\ W^{\left(4n_2+2n_1+n_0\right)\cdot 2k_1} &= \left[W^{8\cdot\left(n_2k_1\right)}\right] \cdot W^{\left(2n_1+n_0\right)\cdot 2k_1} = W^{\left(2n_1+n_0\right)\cdot 2k_1} \end{aligned} \tag{5.53}$$

Damit wird aus (5.51)

$$X\left(n_2, n_1, n_0\right) = \sum_{k_0=0}^{1} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} x_0\left(k_2, k_1, k_0\right) \cdot \mathrm{W}^{4n_0k_2} \cdot \mathrm{W}^{\left(2n_1+n_0\right)\cdot 2k_1} \cdot \mathrm{W}^{\left(4n_2+2n_1+n_0\right)\cdot k_0} \tag{5.54}$$

Mit einer Zerlegung analog zu (5.49) erhält man die entsprechenden Gleichungen für die drei Teilschritte und das abschließende Bit-Reversing

$$\begin{aligned} x_1\left(n_0, k_1, k_0\right) &= \sum_{k_2=0}^{1} x_0\left(k_2, k_1, k_0\right) \cdot W^{4n_0k_2} \\ x_2\left(n_0, n_1, k_0\right) &= \sum_{k_1=0}^{1} x_1\left(n_0, k_1, k_0\right) \cdot W^{\left(2n_1+n_0\right)\cdot 2k_1} \\ x_3\left(n_0, n_1, n_2\right) &= \sum_{k_0=0}^{1} x_2\left(n_0, n_1, k_0\right) \cdot \mathrm{W}^{\left(4n_2+2n_1+n_0\right)\cdot k_0} \\ X\left(n_2, n_1, n_0\right) &= x_3\left(n_0, n_1, n_2\right) \end{aligned} \tag{5.55}$$

Hieraus läßt sich der folgende Signalflußgraph für N=8 konstruieren.

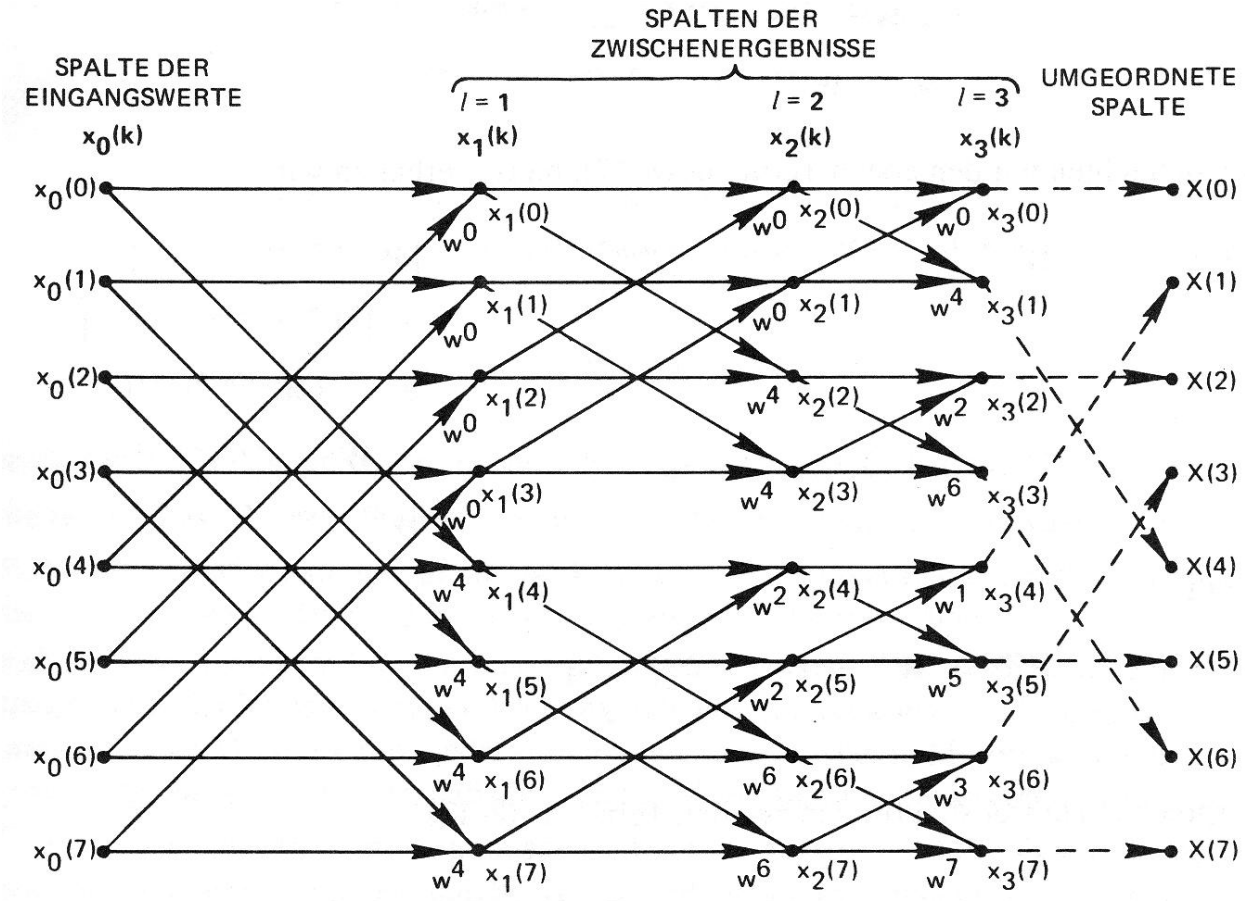


Bild 5.1: Signalflußgraph einer 8-Punkte FFT

5.4 Herleitung des COOLEY-TUKEY-Algorithmus für N=2^g

Hier wird der allgemeine Fall einer DFT mit $N=2^{\gamma}$ Punkten betrachten, wobei γ eine ganze positive Zahl ist. Dann lassen sich *n* und *k* wie folgt als Binärvektoren darstellen:

$$\begin{aligned} n &= 2^{\gamma-1}\cdot n_{\gamma-1} + 2^{\gamma-2}\cdot n_{\gamma-2} + \ldots + n_0 \\ k &= 2^{\gamma-1}\cdot k_{\gamma-1} + 2^{\gamma-2}\cdot k_{\gamma-2} + \ldots + k_0 \end{aligned} \tag{5.56}$$

Damit gilt

$$X\left(n_{\gamma-1}, n_{\gamma-2}, \ldots, n_0\right) = \sum_{k_0=0}^{1}\sum_{k_1=0}^{1}\ldots\sum_{k_{\gamma-1}=0}^{1} x_0\left(k_{\gamma-1}, k_{\gamma-2}, \ldots, k_0\right)\cdot W^{p} \tag{5.57}$$

mit

$$p = \left(2^{\gamma-1}\cdot n_{\gamma-1} + 2^{\gamma-2}\cdot n_{\gamma-2} + \ldots + n_0\right)\cdot\left(2^{\gamma-1}\cdot k_{\gamma-1} + 2^{\gamma-2}\cdot k_{\gamma-2} + \ldots + k_0\right) \tag{5.58}$$

Wegen $W^{a+b} = W^{a}\cdot W^{b}$ kann W^{P} in der Form

$$\begin{aligned} W^{p} = W^{\left(2^{\gamma-1}\cdot n_{\gamma-1}+2^{\gamma-2}\cdot n_{\gamma-2}+\ldots+n_0\right)\cdot\left(2^{\gamma-1}\cdot k_{\gamma-1}\right)} \\ \cdot W^{\left(2^{\gamma-1}\cdot n_{\gamma-1}+2^{\gamma-2}\cdot n_{\gamma-2}+\ldots+n_0\right)\cdot\left(2^{\gamma-2}\cdot k_{\gamma-2}\right)} \ldots \\ \ldots \cdot W^{\left(2^{\gamma-1}\cdot n_{\gamma-1}+2^{\gamma-2}\cdot n_{\gamma-2}+\ldots+n_0\right)\cdot k_0} \end{aligned} \tag{5.59}$$

geschrieben werden. Jetzt wird der erste Term aus (5.59) betrachtet:

$$\begin{aligned} &W^{\left(2^{\gamma-1}\cdot n_{\gamma-1}+2^{\gamma-2}\cdot n_{\gamma-1}+\ldots+n_0\right)\cdot\left(2^{\gamma-1}\cdot k_{\gamma-1}\right)} = \\ &= \underbrace{\left(W^{2^{\gamma}\left(2^{\gamma-2}\cdot n_{\gamma-1}\cdot k_{\gamma-1}\right)}\right)}_{1}\cdot\underbrace{\left(W^{2^{\gamma}\left(2^{\gamma-3}\cdot n_{\gamma-2}\cdot k_{\gamma-1}\right)}\right)}_{1}\cdot\underbrace{\left(W^{2^{\gamma}\left(n_1\cdot k_{\gamma-1}\right)}\right)}_{1}\cdot\underbrace{W^{2^{\gamma-1}\left(n_0\cdot k_{\gamma-1}\right)}}_{\neq 1} = W^{n_0\cdot 2^{\gamma-1}\cdot k_{\gamma-1}} \end{aligned} \tag{5.60}$$

Denn

$$W^{2^{\gamma-1}\cdot 2^{\gamma-1}\cdot(\varepsilon)} = W^{2^{2\gamma-2}\cdot(\varepsilon)} = W^{2^{\gamma}\cdot 2^{\gamma-2}\cdot(\varepsilon)} = \left(W^{N}\right)^{2^{\gamma-2}\cdot(\varepsilon)} = \left(\mathrm{e}^{-\mathrm{j}2\pi\frac{N}{N}}\right)^{2^{\gamma-2}\cdot(\varepsilon)} = 1^{2^{\gamma-2}\cdot(\varepsilon)} = 1$$

Der zweite Term ergibt:

$$W^{\left(2^{\gamma-1}\cdot n_{\gamma-1}+2^{\gamma-2}\cdot n_{\gamma-2}+\ldots+n_0\right)\cdot 2^{\gamma-2}\cdot k_{\gamma-2}} =$$
$$\left[W^{2^{\gamma}\cdot 2^{\gamma-3}\cdot n_{\gamma-1}\cdot k_{\gamma-2}}\right]\cdot\left[W^{2^{\gamma}\cdot 2^{\gamma-4}\cdot n_{\gamma-2}\cdot k_{\gamma-2}}\right]\cdots$$
$$\cdot W^{2^{\gamma-1}\cdot n_1\cdot k_{\gamma-2}}\cdot W^{2^{\gamma-2}\cdot n_0\cdot k_{\gamma-2}} = W^{\left(2n_1+n_0\right)\cdot 2^{\gamma-2}\cdot k_{\gamma-2}} \tag{5.61}$$

Der dritte Term ergibt:

$$W^{\left(4n_2+2n_1+n_0\right)\cdot 2^{\gamma-3}\cdot k_{\gamma-3}} \tag{5.62}$$

Der vierte ergibt:

$$W^{\left(8n_3+4n_2+2n_1+n_0\right)\cdot 2^{\gamma-4}\cdot k_{\gamma-4}} \tag{5.63}$$

Und der letzte:

$$W^{\left(2^{\gamma-1}\cdot n_{\gamma-1}+\ldots+4n_2+2n_1+n_0\right)\cdot k_0} \tag{5.64}$$

Somit hat man schließlich:

$$X\left(n_{\gamma-1},n_{\gamma-2},\ldots,n_0\right)=\sum_{k_0=0}^{1}\sum_{k_1=0}^{1}\ldots\sum_{k_{\gamma-1}=0}^{1}x_0\left(k_{\gamma-1},k_{\gamma-2},\ldots k_0\right)\cdot W^{2^{\gamma-1}\cdot n_0\cdot k_{\gamma-1}}\cdot$$
$$\cdot W^{\left(2n_1+n_o\right)\cdot 2^{\gamma-2}\cdot k_{\gamma-2}}\ldots W^{\left(2^{\gamma-1}\cdot n_{\gamma-1}+2^{\gamma-2}\cdot n_{\gamma-2}+\ldots+n_0\right)\cdot k_0} \tag{5.65}$$

Wenn jetzt die einzelnen Teilsummen Schritt für Schritt berechnet und die Zwischenergebnisse aufgelistet werden, erhält man:

$$x_1\left(n_0,k_{\gamma-2},\ldots,k_0\right)=\sum_{k_{\gamma-1}=0}^{1}x_0\left(k_{\gamma-1},k_{\gamma-2},\ldots,k_0\right)\cdot W^{2^{\gamma-1}\cdot n_0\cdot k_{\gamma-1}}$$

$$x_2\left(n_0,n_1,k_{\gamma-3},\ldots,k_0\right)=\sum_{k_{\gamma-2}=0}^{1}x_1\left(n_0,k_{\gamma-2},\ldots,k_0\right)\cdot W^{\left(2n_1+n_0\right)\cdot 2^{\gamma-2}\cdot k_{\gamma-2}}$$

$$x_3\left(n_0,n_1,n_2,k_{\gamma-4},\ldots,k_0\right)=\sum_{k_{\gamma-3}=0}^{1}x_2\left(n_0,n_1,k_{\gamma-3},\ldots,k_0\right)\cdot W^{\left(4n_2+2n_1+n_0\right)\cdot 2^{\gamma-3}\cdot k_{\gamma-3}} \tag{5.66}$$

$\ldots$

$$x_{\gamma}\left(n_0,n_1,\ldots,n_{\gamma-1}\right)=\sum_{k_0=0}^{1}x_{\gamma-1}\left(n_0,n_1,\ldots,k_0\right)\cdot W^{\left(2^{\gamma-1}\cdot n_{\gamma-1}+2^{\gamma-2}\cdot n_{\gamma-2}+\ldots+n_0\right)\cdot k_0}$$

x_{γ} ergibt sich in Bitumkehrreihenfolge $\Rightarrow$ Reversing ist erforderlich

Dieses sukzessiv abzuarbeitende Gleichungssystem stellt den ursprünglichen Cooley-Tukey-FFT-Algorithmus für $N=2^{\gamma}$ dar. Während die direkte Auswertung einer N-Punkte DFT N^2 komplexe Multiplikationen erfordert, benötigt jede der obigen Gleichungen zwei komplexe Multiplikationen, wobei die erste stets eine ***Multiplikation mit Eins*** ist!

Es gibt insgesamt γ dieser Gleichungen, von denen jede wiederum N Gleichungen repräsentiert. Somit sind insgesamt $N \cdot \gamma$ komplexe Multiplikationen notwendig. Weiterhin tritt bei der Berechnung einer Spalte stets die Beziehung

$$W^{p} = -W^{p+\frac{N}{2}}$$ auf.

Die Anzahl der Multiplikationen reduziert sich damit weiter um den Faktor 1/2. Die Gesamtzahl der komplexen Multiplikationen für $N=2^{\gamma}$ ist somit $N \cdot \gamma/2$. Die Anzahl der komplexen Additionen ist gleich $N \cdot \gamma$.

Anmerkung:

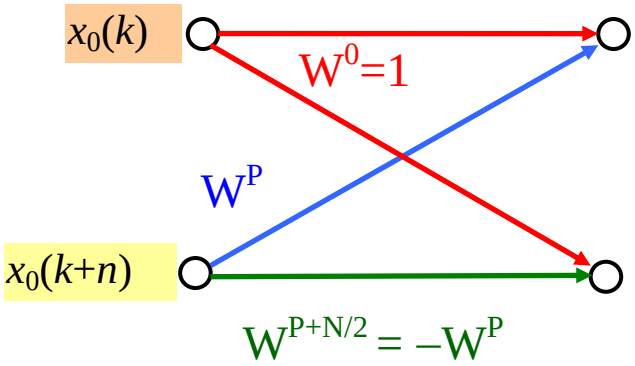
$$W^{p} = -W^{p+\frac{N}{2}}, \quad \text{da} \quad \mathrm{e}^{-\mathrm{j}2\pi\frac{1}{N}\cdot\left(p+\frac{N}{2}\right)} = \mathrm{e}^{-\mathrm{j}2\pi\frac{p}{N}} \cdot \mathrm{e}^{-\mathrm{j}2\pi\frac{N}{2N}} = W^{p} \cdot e^{-\mathrm{j}\pi} = -W^{p}$$

Details eines Beispiels für N=16 (γ=4)

$$X\left(n_3,n_2,n_1,n_0\right)=\sum_{k_0=0}^{1}\sum_{k_1=0}^{1}\sum_{k_2=0}^{1}\sum_{k_3=0}^{1}x_0\left(k_3,k_2,k_1,k_0\right)\cdot$$

$$\cdot W^{n_0\cdot 2^3\cdot k_3}\cdot W^{(2n_1+n_o)\cdot 2^2\cdot k_2}\cdot W^{\left(2^2\cdot n_2+2n_1+n_0\right)2\cdot k_1}\cdot W^{\left(2^3\cdot n_3+2^2\cdot n_2+2n_1+n_0\right)\cdot k_0}$$

n_0, k_2, k_1, k_0	k_3, k_2, k_1, k_0	$(n_0)\cdot 8\cdot k_3$
n_0, n_1, k_1, k_0	n_0, k_2, k_1, k_0	$(2n_1+n_0)\cdot 4\cdot k_2$
n_0, n_1, n_2, k_0	n_0, n_1, k_1, k_0	$(4n_2+2n_1+n_0)\cdot 2\cdot k_1$
n_0, n_1, n_2, n_3	n_0, n_1, n_2, k_0	$(8n_3+4n_2+2n_1+n_0)\cdot k_0$



$x_1(k)=x_0(k)+W^P\cdot x_0(k+n)$

$\Rightarrow x_1(k+n)=2x_0(k)-x_1(k)$

DSP-Befehl:

SUBL $x_1(k),x_0(k)$

$x_1(k+n)=x_0(k)+W^{P+N/2}\cdot x_0(k+n)=x_0(k)-W^P\cdot x_0(k+n)$

SUBL S,D
D:=2D-S
Ergebnis in D

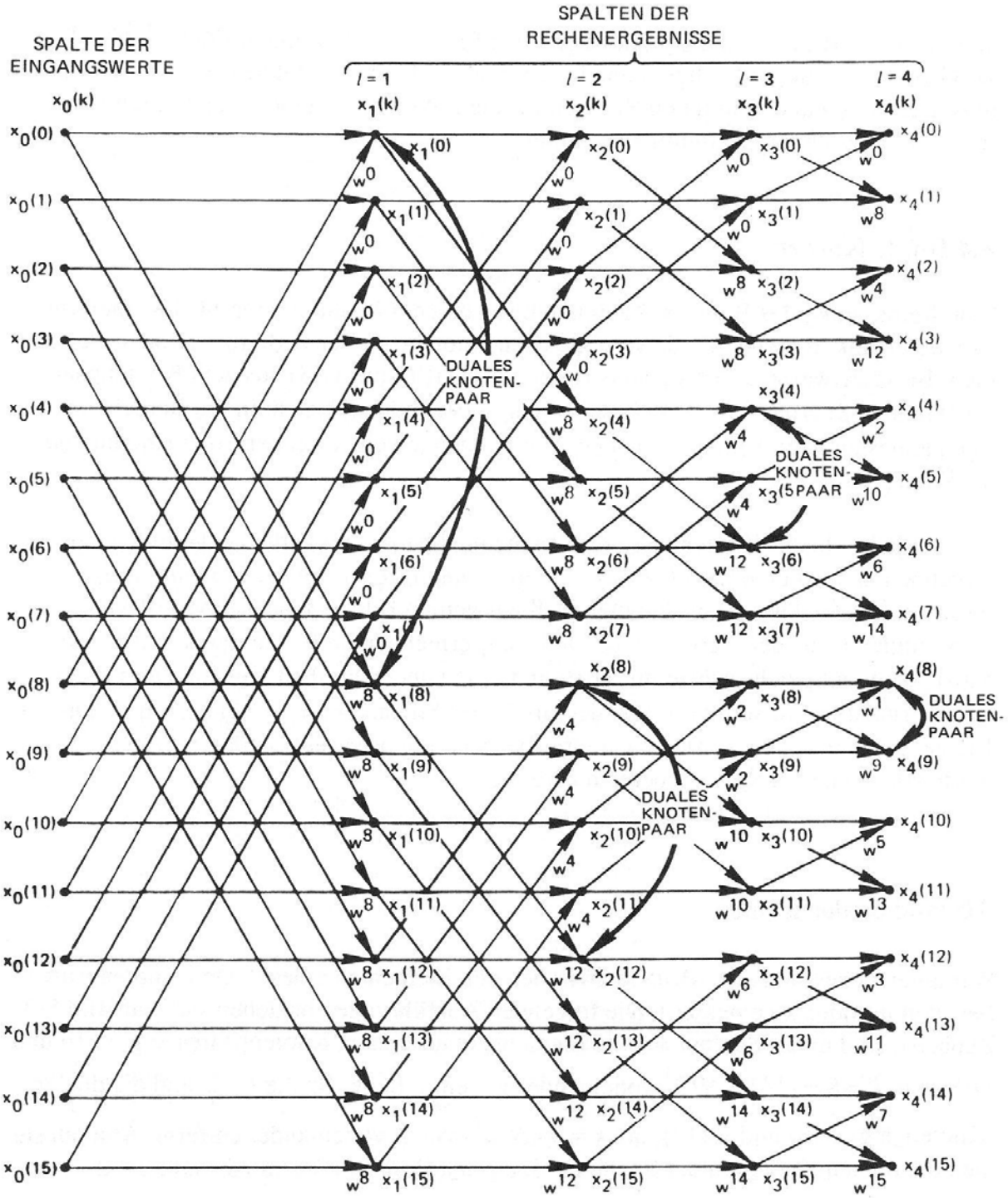


Bild 5.2: Signalflußgraph einer 16-Punkte FFT

5.5 Der Zusammenhang zwischen Fourierreihe und der DFT

Man entwickle die Fourierreihe der in Bild 5.3 dargestellten periodischen Dreieckfunktion $\Lambda(t)$.

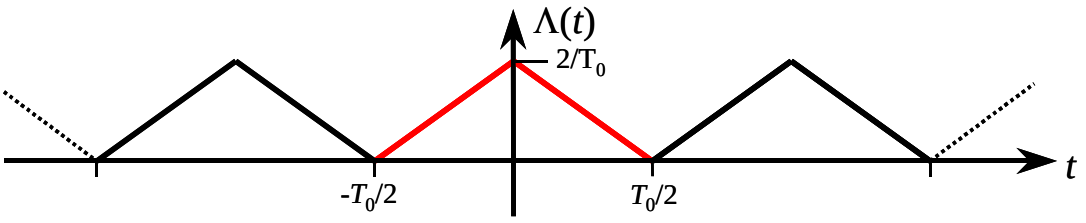


Bild 5.3: Periodische Dreieckfunktion mit Amplitude $2/T_0$ und Frequenz $f_0=1/T_0$

Da $\Lambda(t)$ eine gerade Funktion ist, verschwinden alle Sinusterme und man erhält mit (5.11)

$$\alpha_n = \frac{1}{T_0}\int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} \Lambda(t)\cdot \mathrm{e}^{-\mathrm{j}2\pi n f_0 t}dt = \frac{1}{T_0}\int_{-\frac{T_0}{2}}^{0}\left(\frac{2}{T_0}+\frac{4}{T_0^2}t\right)\cdot\cos\left(2\pi n f_0 t\right)dt + \frac{1}{T_0}\int_{0}^{\frac{T_0}{2}}\left(\frac{2}{T_0}-\frac{4}{T_0^2}t\right)\cdot\cos\left(2\pi n f_0 t\right)dt \quad (5.67)$$

also

$$\alpha_n = \frac{2}{T_0^2}\underbrace{\int_{-T_0/2}^{T_0/2}\cos\left(2\pi n f_0 t\right)dt}_{\frac{1}{2\pi n f_0}\sin(2\pi n f_0 t)\Big|_{-T_0/2}^{T_0/2}} + \frac{4}{T_0^3}\int_{-\frac{T_0}{2}}^{0} t\cdot\cos\left(2\pi n f_0 t\right)dt - \frac{4}{T_0^3}\int_{0}^{\frac{T_0}{2}} t\cdot\cos\left(2\pi n f_0 t\right)dt \quad (5.68)$$

Somit liefert das 1. Integral $\frac{2}{T_0^2}\frac{T_0}{2\pi n}\left[\sin\left(n\pi\right)-\sin\left(-n\pi\right)\right]=0$

Mit $\int x\cdot\cos\left(x\right)dx = \cos\left(x\right)+x\cdot\sin\left(x\right)$ folgt unter Substitution von

$x = 2\pi n f_0 t$ bzw. $t = \frac{x}{2\pi n f_0}$ und damit $dt = \frac{dx}{2\pi n f_0}$ für das 2. Integral

$$\frac{4}{T_0^3}\frac{1}{4\pi^2 n^2 f_0^2}\int_{-\frac{T_0}{2}\cdot 2\pi n f_0}^{0} x\cdot\cos\left(x\right)dx = \frac{1}{T_0\pi^2 n^2}\int_{-n\pi}^{0} x\cdot\cos\left(x\right)dx = \frac{1}{T_0\pi^2 n^2}\left[\cos x + x\sin x\right]_{-n\pi}^{0} =$$

$$\frac{1}{T_0\pi^2 n^2}\left[\underbrace{\cos 0}_{1} - \underbrace{\cos\left(-n\pi\right)}_{\substack{-1 \text{ für ungerade } n\\ +1 \text{ für gerade } n}} + 0 - \left(-n\pi\right)\cdot\underbrace{\sin\left(-n\pi\right)}_{0}\right] = \begin{cases}\frac{2}{T_0\pi^2 n^2} & \text{für ungerade } n\\ 0 & \text{für gerade } n\end{cases} \quad (5.69)$$

und für das 3. Integral ergibt sich unter Verwendung des Teilergebnisses aus (5.69)

$$\frac{1}{T_0\pi^2 n^2}\left[\cos x + x\sin x\right]_{0}^{n\pi} = \frac{1}{T_0\pi^2 n^2}\left[\underbrace{\cos\left(n\pi\right)}_{\substack{-1 \text{ für ungerade } n\\ +1 \text{ für gerade } n}} - \underbrace{\cos 0}_{1} + \left(n\pi\right)\cdot\underbrace{\sin\left(n\pi\right)}_{0} - 0\right] = \begin{cases}\frac{2}{T_0\pi^2 n^2} & \text{für ungerade } n\\ 0 & \text{für gerade } n\end{cases}$$

Somit hat man insgesamt

$$\alpha_n = \begin{cases}\frac{4}{\pi^2\cdot T_0\cdot n^2} & \text{für } n = 1,3,5,\ldots\\ \frac{1}{T_0} & \text{für } n = 0\end{cases}, \quad (5.70)$$

wobei der Fall n=0 noch einer gesonderten Behandlung in (5.68) bedarf

$$\alpha_0 = \frac{2}{T_0^2}\underbrace{\int_{-T_0/2}^{T_0/2}\underbrace{\cos\left(0\right)}_{1}dt}_{T_0} + \frac{4}{T_0^3}\int_{-\frac{T_0}{2}}^{0} t\cdot\cos\left(0\right)dt - \frac{4}{T_0^3}\int_{0}^{\frac{T_0}{2}} t\cdot\cos\left(0\right)dt = \frac{2}{T_0} + \frac{4}{T_0^3}\frac{t^2}{2}\Big|_{-T_0/2}^{0} - \frac{4}{T_0^3}\frac{t^2}{2}\Big|_{0}^{T_0/2}$$

$$\alpha_0 = \frac{2}{T_0} + \frac{2}{T_0^3}\left[0 - \frac{(-T_0)^2}{4} - \frac{(T_0)^2}{4} - 0\right] = \frac{2}{T_0} + \frac{2}{T_0^3}\frac{-T_0^2}{2} = \frac{2}{T_0} - \frac{1}{T_0} = \frac{1}{T_0}$$

Nach (5.10) ergibt sich wegen b_n=0 $\quad \alpha_n = \frac{1}{2}(a_n - \mathrm{j}b_n) \Rightarrow a_n = 2 \cdot \alpha_n$

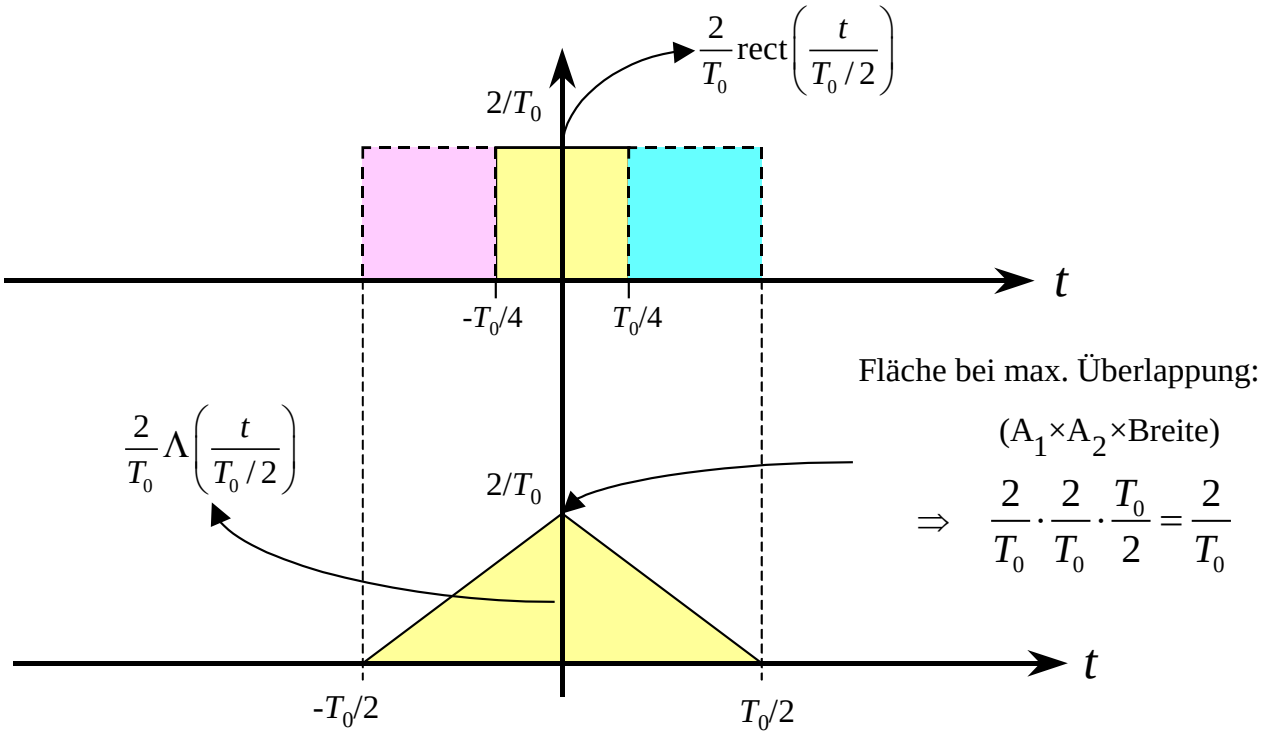
Damit erhält man aus (5.1)

$$\Lambda_p(t) = \frac{1}{T_0} + \frac{8}{\pi^2 \cdot T_0}\left[\cos(2\pi f_0 t) + \frac{1}{3^2}\cos(6\pi f_0 t) + \frac{1}{5^2}\cos(10\pi f_0 t) + \ldots\right] \qquad (5.71)$$

5.5.1 Fourierreihe als Spezialfall des Fourierintegrals

Hier wird eine alternative Berechnung der FT der periodische Dreieckimpulsfolge aus Bild 5.3 betrachtet. Dazu wird zunächst ein Dreieckimpuls mittels Faltung zweier Rechtecke erzeugt.

$$\frac{2}{T_0}\mathrm{rect}\left(\frac{t}{T_0/2}\right) * \frac{2}{T_0}\mathrm{rect}\left(\frac{t}{T_0/2}\right) = \frac{2}{T_0}\Lambda\left(\frac{t}{T_0/2}\right)$$



Fläche bei max. Überlappung:

($A_1 \times A_2 \times$Breite)

$$\Rightarrow \quad \frac{2}{T_0} \cdot \frac{2}{T_0} \cdot \frac{T_0}{2} = \frac{2}{T_0}$$

Bild 5.4: Entstehung einer Dreiecksfunktion durch Faltung von zwei gleichen Rechtecken

Wie man sieht, ergibt sich bei einem Rechteck der Höhe $2/T_0$ und der Breite $T_0/2$ die gleiche Parametrierung der Dreieckfunktion wie in Bild 5.3. Man beachte, daß bei der zur Darstellung der Rechteckfunktion eingeführten der Schreibweise ***rect(t)*** die „Sprungstellen“ jeweils dort liegen, wo das Argument betragsmäßig ½ ist. Bei der Dreieckfunktion Λ(t) gilt hingegen, daß Beginn und Ende der Rampen jeweils dort liegen, wo das Argument den Betrag 1 hat.

Mit der Beziehung $\mathrm{rect}\left(t/T\right) \circ\!\!-\!\!\bullet\, T \cdot \mathrm{si}(\pi f T)$ läßt sich nun in einfacher Weise die FT der Dreieckfunktion bestimmen:

$$\frac{2}{T_0}\mathrm{rect}\left(\frac{t}{T_0/2}\right) * \frac{2}{T_0}\mathrm{rect}\left(\frac{t}{T_0/2}\right) \quad \circ\!\!-\!\!\bullet \quad \frac{2}{T_0} \cdot \frac{T_0}{2}\mathrm{si}\left(\pi f \frac{T_0}{2}\right) \cdot \frac{2}{T_0} \cdot \frac{T_0}{2}\mathrm{si}\left(\pi f \frac{T_0}{2}\right) = \mathrm{si}^2\left(\pi f \frac{T_0}{2}\right) \qquad (5.72)$$

Nun gilt es, aus dem in Bild 5.4 konstruierten Dreieck eine periodische Dreieckfunktion zu generieren. Das läßt sich durch Faltung mit einer unendliche Dirac-Impulsfolge mit dem Impulsabstand T_0, d.h. der Dauer des Dreiecks erreichen. Eine solche Folge ist z.B. durch $\frac{1}{|T_0|}\text{Ш}\left(\frac{t}{T_0}\right)=\sum_{n=-\infty}^{\infty}\delta\left(t-nT_0\right)$ gegeben. Im Zeitbereich führt die Faltung der Dreieckfunktion $\frac{2}{T_0}\Lambda\left(\frac{t}{T_0/2}\right)$ aus Bild 5.4 mit dieser Folge zu dem Ergebnis $\frac{2}{T_0}\Lambda\left(\frac{t}{T_0/2}\right)*\sum_{n=-\infty}^{\infty}\delta\left(t-nT_0\right)=\frac{2}{T_0}\sum_{n=-\infty}^{\infty}\Lambda\left(\frac{t-nT_0}{T_0/2}\right)$

Im weiteren interessiert uns das Spektrum der nunmehr periodische Dreieckimpulsfolge. Da die Faltung im Zeitbereich einer Multiplikation der Spektren entspricht, muß das in (5.72) schon bestimmte Spektrum des einzelnen Dreiecks jetzt noch mit der FT der unendliche Dirac-Impulsfolge multipliziert werden. Diese wurde weiter oben bereits mehrfach verwendet und die Beziehung lautet: $\sum_{n=-\infty}^{\infty}\delta\left(t-nT\right)\circ\!\!-\!\!\bullet\,\frac{1}{|T|}\cdot\sum_{n=-\infty}^{\infty}\delta\left(f-\frac{n}{T}\right)$. Das gesuchte Spektrum der periodischen Dreieckimpulsfolge ergibt sich demnach zu

$$\frac{1}{T_0}\cdot\sum_{n=-\infty}^{\infty}\text{si}^2\left(\pi\frac{n}{T_0}\frac{T_0}{2}\right)\cdot\delta\left(f-\frac{n}{T_0}\right)=\frac{1}{T_0}\cdot\sum_{n=-\infty}^{\infty}\text{si}^2\left(\frac{n\pi}{2}\right)\cdot\delta\left(f-\frac{n}{T_0}\right). \qquad (5.73)$$

Wie im weiteren noch klar wird, repräsentiert (5.73) nichts anderes als die komplexen Fourierkoeffizienten α_n wie sie in (5.70) bestimmt wurden. Dies soll an einigen Beispielwerten kurz verifiziert werden. Für n=0 erhält man sofort $\alpha_0=1/T_0$

Bei der Darstellung des gesamten Spektrums ist zu beachten, daß für jeden Wert von n auch das korrespondierende Resultat für negatives n berücksichtigt werden muß. Da wir eine gerade Funktion vor uns haben, genügt das Berechnen für positives n. Die Ergebnisse können dann einfach für $-n$ kopiert werden. Für n=1 erhalten wir in Übereinstimmung mit (5.70) an der Stelle $f=1/T_0$ $\alpha_1=\frac{1}{T_0}\cdot\text{si}^2\left(\frac{\pi}{2}\right)=\frac{1}{T_0}\frac{\sin^2\left(\pi/2\right)}{\pi^2/4}=\frac{4}{T_0\pi^2}$. Für n=2 würde sich $\alpha_2=\frac{1}{T_0}\cdot\text{si}^2\left(\frac{2\pi}{2}\right)\cdot\delta\left(f-\frac{2}{T_0}\right)=0$ ergeben, was im übrigen für alle geraden Werte von n gilt, so daß nur ungerade n zu berücksichtigen sind. Ein letztes Beispiel zeigt, daß auch bei Einsetzen von n=3 mit $\alpha_3=\frac{1}{T_0}\frac{\sin^2\left(3\pi/2\right)}{9\pi^2/4}=\frac{4}{9\cdot T_0\cdot\pi^2}$ die exakte Übereinstimmung mit (5.70) gegeben ist. In den Grafiken von Bild 5.5 sind die diskutierten Sachverhalte noch mal zusammenfassend und anschaulich dargestellt.

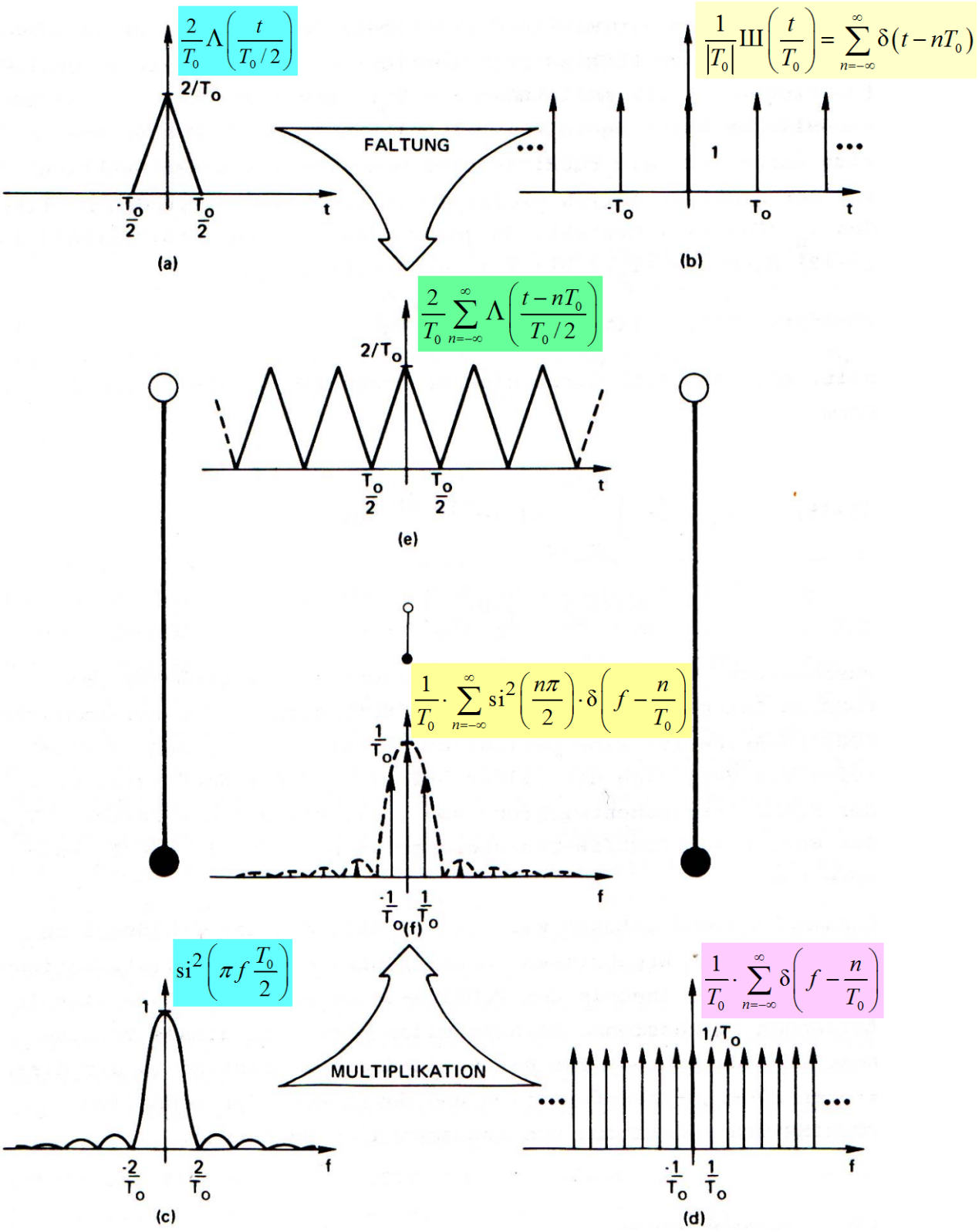


Bild 5.5: Anschauliche Herleitung der Fouriertransformierten einer periodischen Dreieckfunktion unter Anwendung des Faltungstheorems

5.5.2 Häufig benötigte Beziehungen und Fouriertransformationspaare

Bevor wir zu einer genauen Untersuchung des Übergangs von der kontinuierlichen FT zur DFT kommen, sind in folgenden einige wichtige Beziehungen der FT und Transformationspaare zusammengestellt, die man beim Arbeiten mit der Systemtheorie parat haben sollte.

Skalierung bei der FT: $$h(\mathrm{k}t) \quad \circ\!\!-\!\!\bullet \quad \frac{1}{|\mathrm{k}|}\cdot H\left(\frac{f}{\mathrm{k}}\right)$$

Bei Skalierung von Deltafunktionen gilt: $$\delta(\mathrm{a}t) = \frac{1}{|\mathrm{a}|}\delta(t)$$

Symmetrie: $$H(t) \quad \circ\!\!-\!\!\bullet \quad h(-f)$$

Zeitverschiebung: $$h(t-t_0) \circ\!\!-\!\!\bullet H(f)\cdot \mathrm{e}^{-\mathrm{j}2\pi f t_0}$$

Frequenzverschiebung (Modulation) $$h(t)\cdot \mathrm{e}^{\mathrm{j}2\pi f_0 t} \circ\!\!-\!\!\bullet H(f-f_0)$$

Rechteckimpuls

$$A\cdot \mathrm{rect}\left(\frac{t}{T_0}\right) \quad \circ\!\!-\!\!\bullet \quad AT_0\cdot \mathrm{si}(\pi T_0 f)$$

Dreieckimpuls

$$A_r\cdot \mathrm{rect}\left(\frac{t}{T_0/2}\right) * A_r\cdot \mathrm{rect}\left(\frac{t}{T_0/2}\right) = A_r^2\cdot\frac{T_0}{2}\cdot\Lambda\left(\frac{t}{T_0/2}\right)$$

$$\circ\!\!-\!\!\bullet A_r\cdot\frac{T_0}{2}\mathrm{si}\left(\pi f\frac{T_0}{2}\right)\cdot A_r\cdot\frac{T_0}{2}\mathrm{si}\left(\pi f\frac{T_0}{2}\right) = A_r^2\cdot\frac{T_0^2}{4}\mathrm{si}^2\left(\pi f\frac{T_0}{2}\right)$$

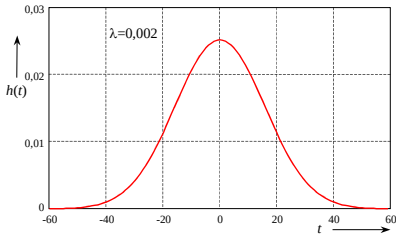
Folge äquidistanter Diracimpulse (normiert)

$$\mathrm{Ш}(t) = \sum_{n=-\infty}^{\infty}\delta(t-n) \quad \circ\!\!-\!\!\bullet \quad \mathrm{Ш}(f) = \sum_{n=-\infty}^{\infty}\delta(f-n)$$

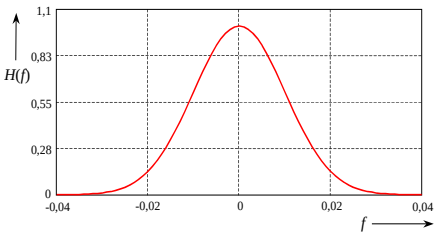
Folge äquidistanter Diracimpulse (skaliert)

$$\frac{1}{|T|}\mathrm{Ш}\left(\frac{t}{T}\right) = \sum_{n=-\infty}^{\infty}\delta(t-nT) \circ\!\!-\!\!\bullet \frac{1}{|T|}\cdot\sum_{n=-\infty}^{\infty}\delta\left(f-\frac{n}{T}\right)$$

Gaußsignal



$$\sqrt{\frac{\lambda}{\pi}}\cdot \mathrm{e}^{-\lambda t^2} \circ\!\!-\!\!\bullet \quad \mathrm{e}^{\frac{-\pi^2\cdot f^2}{\lambda}}$$



Cosinussignal (ohne Zeitbegrenzung)

$$A\cos(2\pi f_0 t) \quad \circ\!\!-\!\!\bullet \quad \frac{A}{2}\left[\delta(f-f_0)+\delta(f+f_0)\right]$$

Sinussignal (ohne Zeitbegrenzung)

$$A\cdot\sin(2\pi f_0 t) \quad \circ\!\!-\!\!\bullet \quad \mathrm{j}\frac{A}{2}\left[\delta(f+f_0)-\delta(f-f_0)\right]$$

Cosinussignal mit Zeitbegrenzung

$$A\cdot\mathrm{rect}\left(\frac{t}{nT_0}\right)\cos(2\pi f_0 t) \quad \circ\!\!-\!\!\bullet \quad \frac{A\cdot nT_0}{2}\cdot\mathrm{si}(\pi nT_0 f)*\left[\delta(f-f_0)+\delta(f+f_0)\right]$$

$$=\frac{A\cdot nT_0}{2}\cdot\left\{\mathrm{si}\left[\pi nT_0(f-f_0)\right]+\mathrm{si}\left[\pi nT_0(f+f_0)\right]\right\}$$

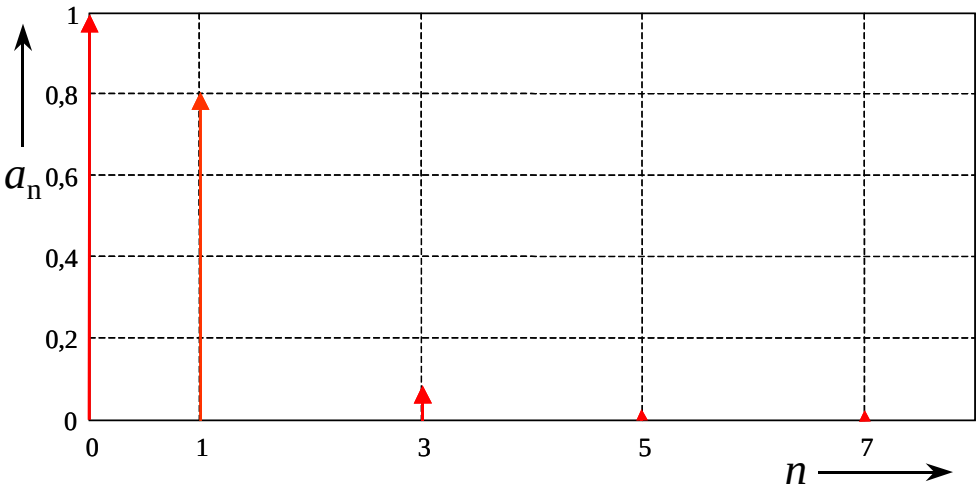
Summe von Cosinussignalen mit gleicher Zeitbegrenzung (Grundlage für OFDM)

$$\sum_{i=1}^{N} A\cdot\mathrm{rect}\left(\frac{t}{nT_0}\right)\cos(2\pi\cdot i\cdot f_0 t) \circ\!\!-\!\!\bullet \frac{A\cdot nT_0}{2}\cdot\sum_{i=1}^{N}\mathrm{si}\left[\pi nT_0(f-if_0)\right]+\mathrm{si}\left[\pi nT_0(f+if_0)\right]$$

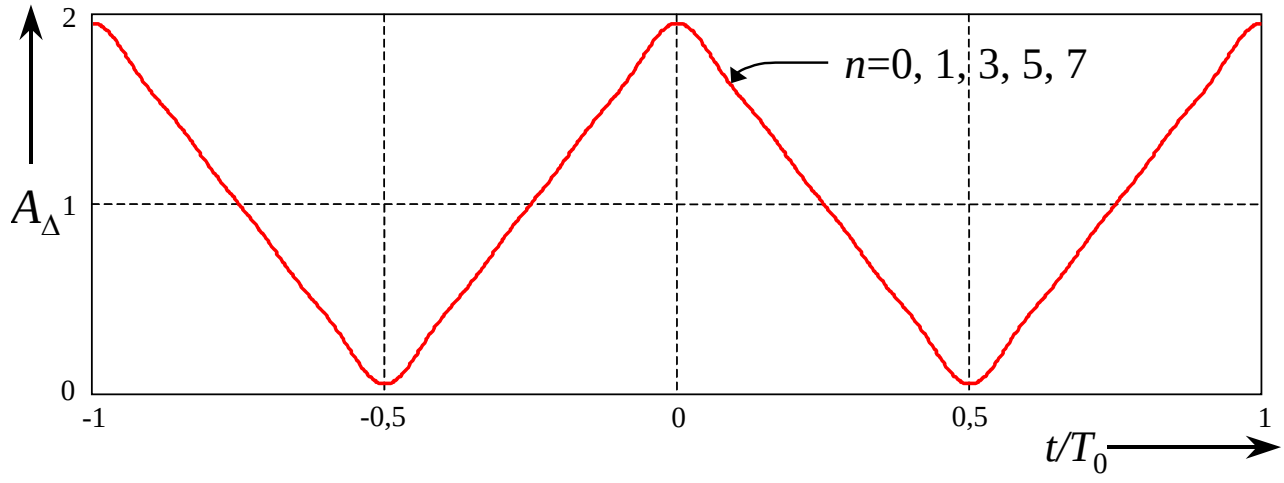
Beispiele für Fourierreihen

Periodische Dreieckimpulsfolge

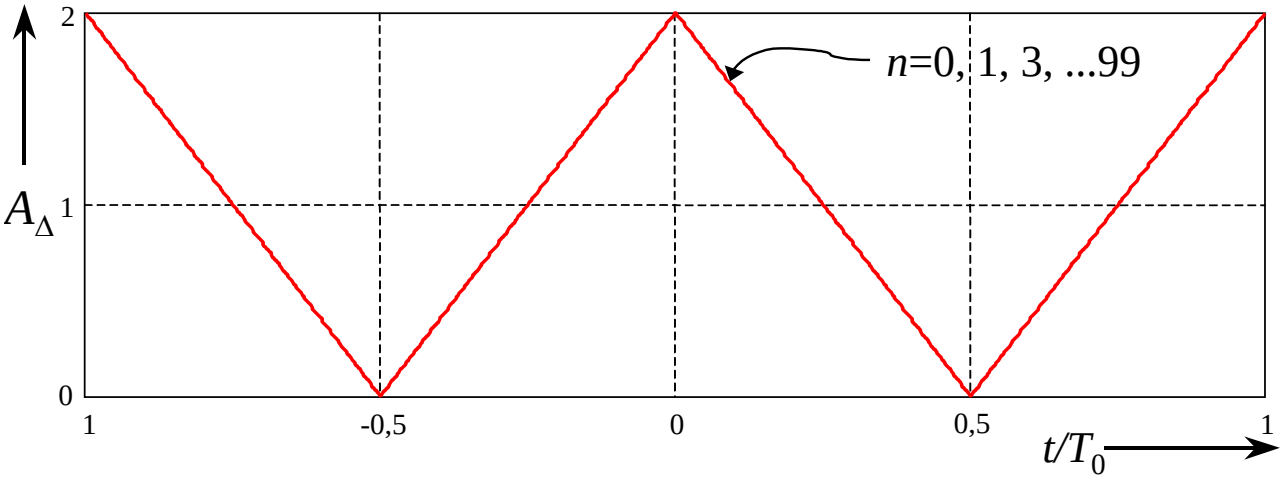
$$\frac{2}{T_0}\sum_{n=-\infty}^{\infty}\Lambda\left(\frac{t-nT_0}{T_0/2}\right) \circ\!\!-\!\!\bullet \frac{1}{T_0}\cdot\sum_{n=-\infty}^{\infty}\mathrm{si}^2\left(\frac{n\pi}{2}\right)\cdot\delta\left(f-\frac{n}{T_0}\right)$$



Im Bild links sind die ersten 7 Koeffizienten a_n der Fourierreihe, die die periodische Dreieckfunktion nach Bild 5.3 beschreiben, dargestellt. Im untenstehenden Bild sieht man das zugehörige Syntheseergebnis, das die Dreieckfunktion schon sehr gut annähert.



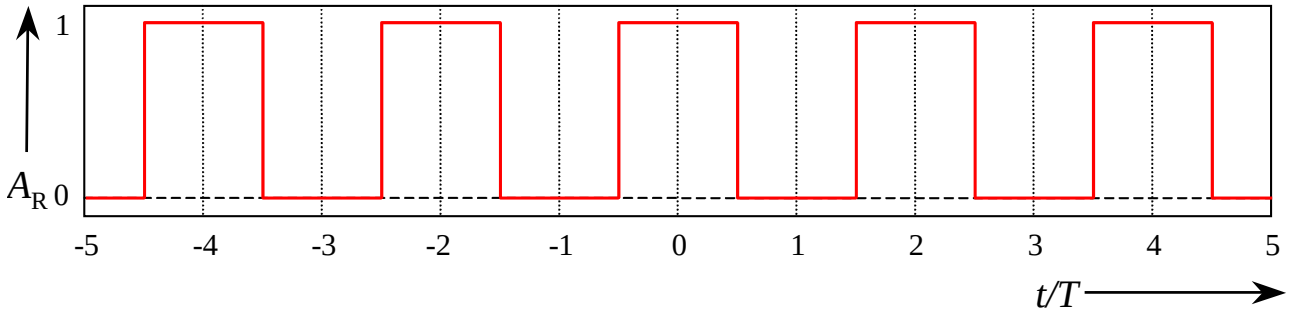
Wenn man z.B. alle Koeffizienten bis a_{99} berücksichtigt, ergibt sich die folgende perfekte Darstellung der Dreieckfunktion.

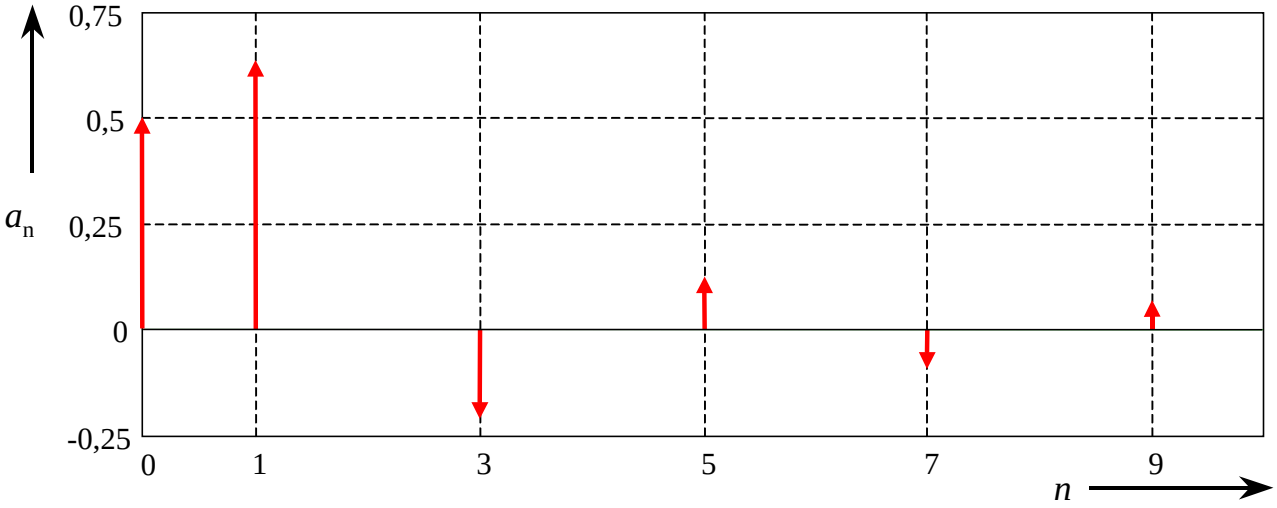


Periodische Rechteckimpulsfolge

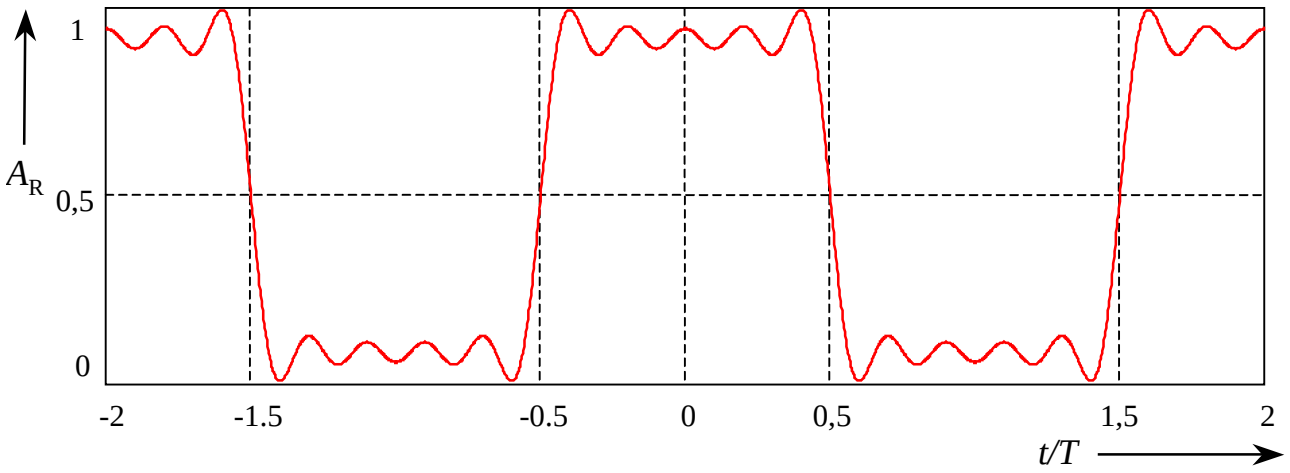
$$A \cdot \mathrm{rect}\left(\frac{t}{T}\right) * \sum_{n=-\infty}^{\infty} \delta(t - 2nT) \circ\!\!-\!\!\bullet$$

$$AT \cdot \mathrm{si}(\pi fT) \cdot \frac{1}{2T} \cdot \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{2T}\right) = A \sum_{n=-\infty}^{\infty} \mathrm{si}\left(\frac{n\pi}{2}\right) \cdot \delta\left(f - \frac{n}{2T}\right)$$

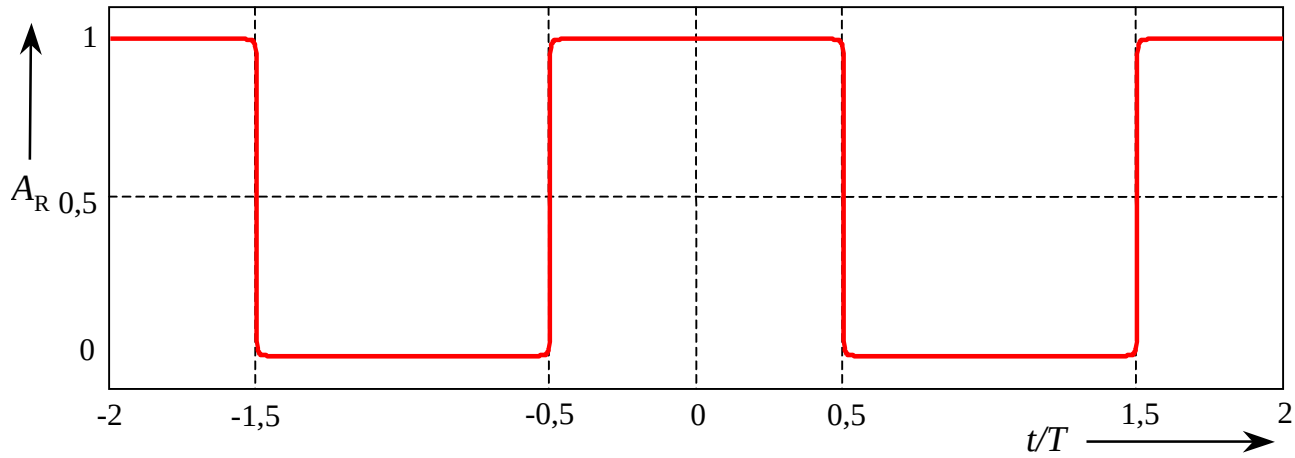




Koeffizienten a_0 .... a_9.der Fourierreihe



Rechteckfolgen-Synthese mit den obigen 9 Koeffizienten der Fourierreihe



Rechteckfolgen-Synthese mit den Koeffizienten der Fourierreihe bis a_{99}

5.6 Zusammenhang zwischen kontinuierlicher FT und DFT

Bei Beschreibungen der DFT wird üblicherweise von einer selbständigen Definition der FT diskreter Signale endlicher Dauer ausgegangen. Aus dieser axiomatischen Definition werden dann die Eigenschaften der DFT abgeleitet. Dieser Beschreibungsweg ungünstig, da am Ende immer die wichtige Frage offen bleibt, welcher Zusammenhang zwischen der DFT der kontinuierlichen FT wirklich besteht. Um diese Frage zu beantworten, erweist es sich als vorteilhaft, die DFT als „Sonderfall“ aus der kontinuierlichen FT herzuleiten.

In den folgenden Beschreibungen der FT wird eine Form gewählt, die sich für die Implementierung auf PCs oder digitalen Signalprozessoren besonders eignet. Die DFT wird mit graphischer Unterstützung anschaulich aus der Theorie der kontinuierlichen FT hergeleitet. Beide Darstellungsweisen, die graphische wie auch die mathematische, sind so ausgerichtet, daß die Ergebnisse sich „computergerecht“ darstellen lassen.

Als Beispiel wird im weiteren die Funktion $h(t)$ und ihre Fouriertransformierte $H(f)$ aus Bild 5.6 betrachtet. Dieses Transformationspaar ist jetzt derart zu modifizieren, daß seine Berechnung mit Hilfe eines Digitalrechners auf einfache Art möglich wird. Eine solche Modifikation führt auf die DFT, mit der die kontinuierliche FT so gut wie möglich approximiert werden soll.

Um die FT von $h(t)$ mit Methoden der digitalen Signalverarbeitung zu bestimmen, ist zunächst eine Abtastung nötig, d.h. eine Multiplikation von $h(t)$ mit der bekannten Zeitbereichs-Abtastfunktion

$$\sum_{n=-\infty}^{\infty} \delta(t-nT) = \frac{1}{T}\,\text{Ш}\left(\frac{t}{T}\right) .$$

Das Ergebnis der Abtastung ist ein zeitdiskretes Signal der Form

$$h_{\text{A}}(t) = h(t)\cdot \sum_{n=-\infty}^{\infty} \delta(t-nT) = \sum_{k=-\infty}^{\infty} h(k\text{T})\cdot\delta(t-k\text{T}) \tag{5.74}$$

Da die Multiplikation im Zeitbereich eine Faltung der entsprechenden Fouriertransformierten zur Folge hat, gilt nun für das Spektrum mit $\sum_{n=-\infty}^{\infty} \delta(t-nT) \circ\!\!-\!\!\bullet \frac{1}{T}\sum_{n=-\infty}^{\infty} \delta\left(f-\frac{n}{T}\right)$

$$H_{\text{A}}(f) = H(f) * \frac{1}{T}\sum_{n=-\infty}^{\infty} \delta\left(f-\frac{n}{T}\right) = \frac{1}{T}\sum_{n=-\infty}^{\infty} H\left(f-\frac{n}{T}\right). \tag{5.75}$$

Man beachte, daß die bisherigen Modifikationen außer dem Aliasing bei $H_{\text{A}}(f)$ keine weiteren Abweichungen zwischen FT und DFT bewirkt haben. Um den aus Aliasing resultierenden Fehler zu reduzieren, bleibt nur die Möglichkeit, die Abtastfrequenz zu erhöhen, d.h. T zu verkleinern.

Das Transformationspaar unter (c) in Bild 5.6 ist für eine numerische Auswertung noch nicht geeignet, weil weder im Zeit- noch im Frequenzbereich die Zahl der Abtastwerte endlich ist.

Es ist daher erforderlich, das abgetastete Signal $h_{\text{A}}(t)$ zeitlich zu begrenzen, so daß sich eine endliche Anzahl N von Abtastwerten ergibt. Man erreicht das z.B. durch Einführen einer rechteckförmigen Begrenzungsfunktion rect(·) (Fensterfunktion) mit folgender Parametrierung

$$w(t) = \text{rect}\left(\frac{t - T_0/2 + T/2}{T_0}\right), \tag{5.76}$$

deren Fouriertransformierte

$$W(f) = T_0 \cdot \text{si}(\pi f T_0)\cdot \text{e}^{-\text{j}2\pi f (T_0/2 - T/2)} \quad \text{ist.} \tag{5.77}$$

Die Zeitbegrenzung stellt eine zweite Modifikation des ursprünglichen kontinuierlichen FT-Paares dar. Da hierzu im Zeitbereich eine Multiplikation erfolgte, ergibt sich im Frequenzbereich eine Faltung der entsprechenden Spektren.

Das Spektrum unter (e) weist nun eine gewisse Welligkeit auf. Der Effekt ist zur Verdeutlichung verstärkt dargestellt. Zur quantitativen Analyse sei an die reziprok-proportionale Beziehung zwischen Breite einer Zeitfunktion und der Breite ihrer Fouriertransformierten erinnert. Demnach strebt die si(·)-Funktion, die $W(f)$ charakterisiert, mit einer Verbreiterung der Begrenzungsfunktion $w(t)$ gegen einen Diracimpuls. Je mehr sich $W(f)$ dem Diracimpuls nähert, um so geringer ist die Welligkeit und um so kleiner der Fehler, der durch die Zeitbegrenzung verursacht wird. Prinzipiell wäre es daher wünschenswert, die Breite der Begrenzungsfunktion stets möglichst groß zu wählen. Allerdings steht dem die Erhöhung der Zahl der Abtastwerte und die damit wachsende Rechenzeit entgegen. Zu den unter (e) dargestellten Grafiken gehört also die folgende Beziehung:

$$h_{\text{A}}(t)\cdot w(t) \circ\!\!-\!\!\bullet\ \frac{1}{T}\sum_{n=-\infty}^{\infty} H\left(f-\frac{n}{T}\right) * W(f) \tag{5.78}$$

Dieses Ergebnis ist jedoch noch nicht das gewünschte DFT-Paar, weil im Frequenzbereich immer noch eine kontinuierliche Funktion vorliegt, die für eine numerische Behandlung mit einer endli-

chen Anzahl diskreter Abtastwerte zu beschreiben ist. Dies geschieht im weiteren durch Abtastung im Frequenzbereich mit Hilfe der unter (f) [rechts] skizzierten Funktion

$$T_0 \cdot \text{Ш}\left(fT_0\right) = \sum_{r=-\infty}^{\infty} \delta\left(f - \frac{r}{T_0}\right) \quad \bullet\!\!-\!\!\circ \quad \text{Ш}\left(\frac{t}{T_0}\right) = T_0 \sum_{r=-\infty}^{\infty} \delta\left(t - rT_0\right). \quad (5.79)$$

Dieser letzte Modifizierungsschritt liefert das gesuchte DFT-Paar. Im Zeitbereich entspricht die Abtastung des Spektrums der Faltung des zeitbegrenzten Abtastsignals nach (5.78) mit der Zeitfunktion $\text{Ш}\left(t/T_0\right)$ aus (5.79), so daß nunmehr unter (g) [links] eine mit f_0=1T_0 periodische Zeitfunktion

$$\tilde{h}\left(t\right) = h_{\text{A}}\left(t\right) \cdot w\left(t\right) * \text{Ш}\left(t/T_0\right) = \left[\sum_{k=0}^{N-1} h\left(kT\right) \cdot \delta\left(t - kT\right)\right] * T_0 \sum_{r=-\infty}^{\infty} \delta\left(t - rT_0\right) \quad (5.80)$$

vorliegt, der im Frequenzbereich das ebenfalls periodische diskrete Spektrum

$$\widetilde{H}\left(f\right) = \left[\frac{1}{T}\sum_{n=-\infty}^{\infty} H\left(f - \frac{n}{T}\right) * W\left(f\right)\right] \cdot \sum_{r=-\infty}^{\infty} \delta\left(f - \frac{r}{T_0}\right) \quad (5.81)$$

entspricht – s. Bild 5.6 (g) [rechts]. Das nun erzielte DFT-Paar eignet sich für die numerische Behandlung, da sowohl die Zeit- als auch die Frequenzfunktion in diskreter Form vorliegen, d.h. die ursprüngliche Zeitfunktion $h(t)$ ist ebenso wie die ursprüngliche Fouriertransformierte $H(f)$ durch N Abtastwerte approximiert. Die beiden Folgen von je N Abtastwerten bilden ein DFT-Paar und sind eine Approximation des ursprünglichen kontinuierlichen Transformationspaares.

Man beachte, daß eine Abtastung im Zeitbereich zu einer periodischen Frequenzfunktion und eine Abtastung im Frequenzbereich zu einer periodischen Zeitfunktion führt. Demnach verlangt die DFT, daß sowohl die ursprüngliche Zeit- als auch die Frequenzfunktion so zu modifizieren sind, daß aus ihnen periodische Funktionen entstehen. N Zeitabtastwerte repräsentieren dann eine Zeitperiode und N Frequenzabtastwerte eine Frequenzperiode. Da die Werte aus dem Zeit- und aus dem Frequenzbereich durch die kontinuierliche FT miteinander in Zusammenhang stehen, lassen sich stets eindeutige Beziehungen herleiten.

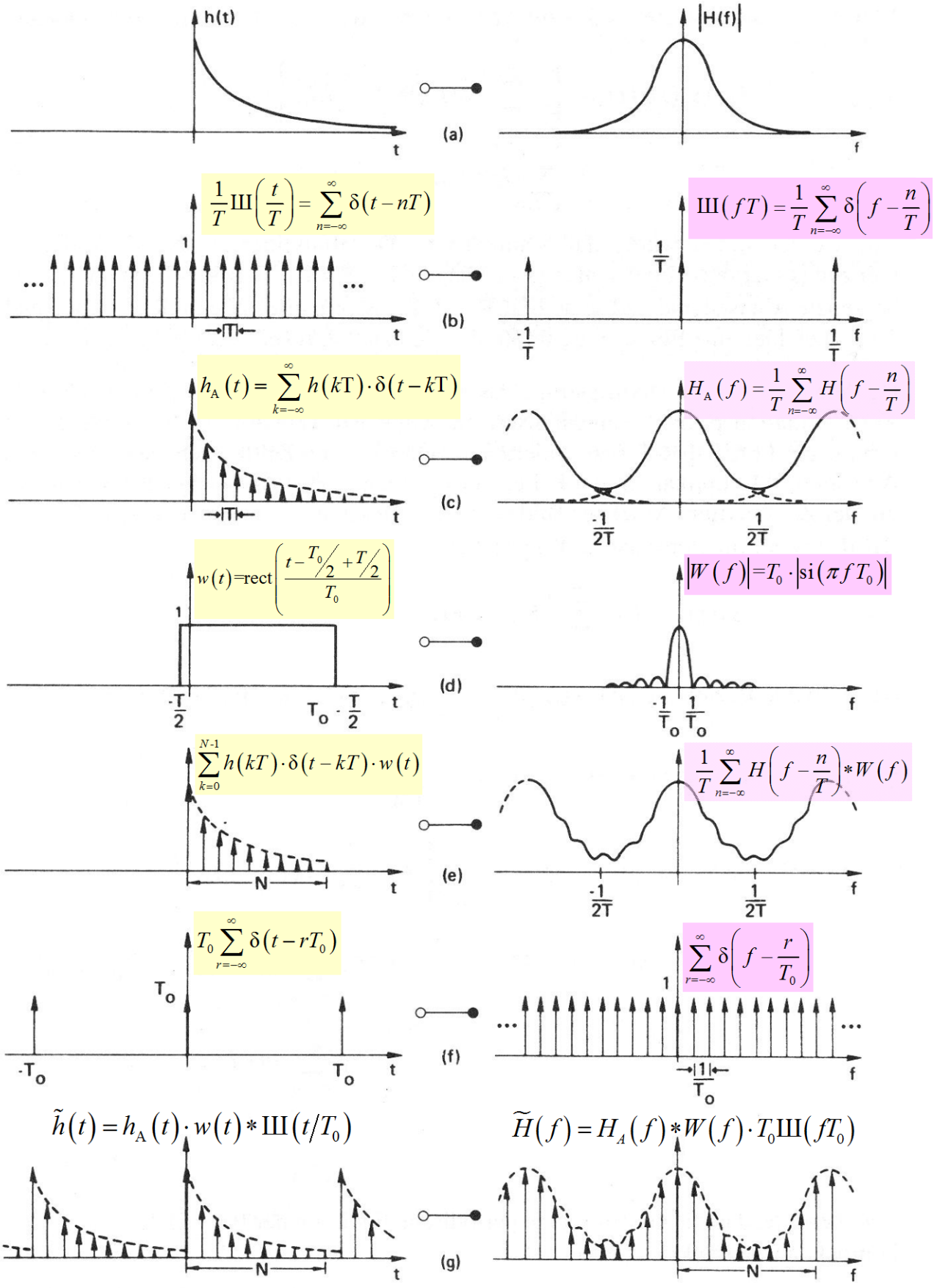


Bild 5.6: Anschauliche Beschreibung des Übergangs von der kontinuierlichen FT zur DFT

Die Ausführung der Faltung in (5.80) ergibt

$$\tilde{h}(t) = T_0 \sum_{r=-\infty}^{\infty} \left[\sum_{k=0}^{N-1} h(kT) \cdot \delta(t - kT - rT_0) \right]. \tag{5.82}$$

Eine „Entwicklung“ von (5.82) um r=0 herum (z.B. r=–1, r=0, r=1) liefert die Terme

$$...T_0 \sum_{k=0}^{N-1} h(kT) \cdot \delta\underbrace{(t + T_0 - kT)}_{r=-1} + T_0 \sum_{k=0}^{N-1} h(kT) \cdot \delta\underbrace{(t - kT)}_{r=0} + T_0 \sum_{k=0}^{N-1} h(kT) \cdot \delta\underbrace{(t - T_0 - kT)}_{r=1} + ...$$

Man sieht, daß (5.82) eine periodische Funktion mit der Periodendauer T_0 beschreibt.

Das Symbol ~ über $h(t)$ deutet an, daß eine Approximation vorliegt, die mehr oder weniger fehlerbehaftet sein kann.

Die spezielle Wahl der Fensterfunktion $w(t)$ in (5.76) bedarf noch einer kurzen Erläuterung. Man sieht, daß das Faltungsprodukt aus (5.82) eine periodische Funktion mit der Periodendauer T_0 ist, die innerhalb einer Periode genau N Abtastwerte enthält. Wenn die rechteckförmige Fensterfunktion $w(t)$ nun so gewählt wäre, daß jeweils eine Flanke mit einem Abtastwert zusammenfiele, dann hätte die Faltung von $w(t)$ mit den Diracimpulsen im Abstand T_0 eine Überlappung im Zeitbereich zur Folge, d.h. der N-te Abtastwert einer jeden Periode fiele mit dem ersten Abtastwert der darauffolgenden zusammen. Eine geringfügige Verschiebung des Fensters, z.B. um eine halbe Abtastdauer, löst das Problem auf einfache Weise.

Zur Bestimmung der Fouriertransformierten von (5.82) greifen wir auf die exponentielle Darstellung der Fourierreihe gemäß (5.12) zurück, deren FT sich unmittelbar mit Hilfe des Modulationstheorems (5.23) $h(k) \cdot \mathrm{e}^{\mathrm{j}2\pi ik/N} \circ\!\!-\!\!\bullet\, H(n-i)$ bestimmen läßt. Da die Koeffizienten α_n nicht zeitabhängig, d.h. Konstanten sind, liefert ihre FT gewichtete Diracimpulse, so daß sich

$$h(t) = \sum_{n=-\infty}^{\infty} \alpha_n \cdot \mathrm{e}^{\mathrm{j}2\pi n f_0 t} \circ\!\!-\!\!\bullet\, H(f) = \sum_{n=-\infty}^{\infty} \alpha_n \cdot \delta(f - nf_0) \tag{5.83}$$

ergibt, wobei hier

$$\alpha_n = \frac{1}{T_o} \int_{-\frac{T}{2}}^{T_o - \frac{T}{2}} \tilde{h}(t) \cdot \mathrm{e}^{-\mathrm{j}2\pi nt/T_0} dt \;, \quad \text{mit} \quad n = 0,\ \pm 1,\ \pm 2, ... \tag{5.84}$$

ist. Das Einsetzen von (5.82) in (5.84) liefert

$$\alpha_n = \frac{1}{T_0} \int_{-\frac{T}{2}}^{T_0 - \frac{T}{2}} T_0 \sum_{r=-\infty}^{\infty} \sum_{k=0}^{N-1} h(kT) \cdot \delta(t - kT - rT_0) \cdot \mathrm{e}^{-\mathrm{j}2\pi nt/T_0} dt \,. \tag{5.85}$$

Da die Integration sich nur über eine Periode erstreckt, ist in (5.85) r=0 zu setzen und man erhält

$$\begin{aligned} \alpha_n &= \int_{-\frac{T}{2}}^{T_0 - \frac{T}{2}} \sum_{k=0}^{N-1} h(kT) \cdot \delta(t - kT) \cdot \mathrm{e}^{-\mathrm{j}2\pi nt/T_0} dt = \\ &= \sum_{k=0}^{N-1} h(kT) \int_{-\frac{T}{2}}^{T_0 - \frac{T}{2}} \mathrm{e}^{-\mathrm{j}2\pi nt/T_0} \cdot \delta(t - kT)\, dt = \sum_{k=0}^{N-1} h(kT) \cdot \mathrm{e}^{-\mathrm{j}2\pi (n \cdot k) T/T_0} \end{aligned} \tag{5.86}$$

Wenn – wie in Bild 5.6 vorausgesetzt – T_0=NT ist, folgt aus (5.86)

$$\alpha_n = \sum_{k=0}^{N-1} h(kT) \cdot \mathrm{e}^{-\mathrm{j}2\pi(k\cdot n)/N}, \quad \text{mit} \quad n = 0, \ \pm 1, \ \pm 2, \ldots, \tag{5.87}$$

so daß die FT von (5.82) unter Verwendung von (5.83)

$$H(f) = \sum_{n=-\infty}^{\infty} \tilde{H}\left(\frac{n}{NT}\right) \cdot \delta(f - nf_o) \ \text{ mit } \ \tilde{H}\left(\frac{n}{NT}\right) = \sum_{k=0}^{N-1} h(kT) \cdot \mathrm{e}^{-\mathrm{j}2\pi(k\cdot n)/N} \tag{5.88}$$

ergibt. Aus einer flüchtigen Betrachtung von (5.88) kann man nicht sofort erkennen, daß die FT $H(f)$, wie sie in Bild 5.6 unter (g) dargestellt ist, periodisch ist. Denn aus $\tilde{H}(n/NT)$ lassen sich nur N diskrete komplexe Werte errechnen. Um diesen Punkt zu verdeutlichen, setze man $n=r$, wobei r eine beliebige ganze Zahl ist. Dann ergibt sich

$$\tilde{H}\left(\frac{r}{NT}\right) = \sum_{k=0}^{N-1} h(kT) \cdot \mathrm{e}^{-\mathrm{j}2\pi(k\cdot r)/N} \tag{5.89}$$

Jetzt sei $n=r+N$. Wegen $\mathrm{e}^{-\mathrm{j}2\pi k} = \cos(2\pi k) - \mathrm{j} \cdot \sin(2\pi k) = 1$ gilt für alle ganzzahligen k:

$$\mathrm{e}^{-\mathrm{j}2\pi k(r+N)/N} = \mathrm{e}^{-\mathrm{j}2\pi kr/N} \cdot \mathrm{e}^{-\mathrm{j}2\pi k} = \mathrm{e}^{-\mathrm{j}2\pi kr/N} \tag{5.90}$$

Damit erhält man für $n=r+N$

$$\tilde{H}\left(\frac{r+N}{NT}\right) = \sum_{k=0}^{N-1} h(kT) \cdot \mathrm{e}^{-\mathrm{j}2\pi k(r+N)/N} = \sum_{k=0}^{N-1} h(kT) \cdot \mathrm{e}^{-\mathrm{j}2\pi kr/N} = \tilde{H}\left(\frac{r}{NT}\right) \tag{5.91}$$

Folglich lassen sich aus (5.88) nur N diskrete Werte errechnen. $\tilde{H}(n/NT)$ ist also periodisch mit der Periode N. (5.88) beschreibt also die gesuchte diskrete Fouriertransformation, die N Abtastwerte einer Zeitfunktion mit N Abtastwerten einer Frequenzfunktion verknüpft. Die DFT ist also ein ***Spezialfall*** der kontinuierlichen FT. Wenn die N Abtastwerte einer ursprünglichen kontinuierlichen Funktion $h(t)$ als Periode einer (künstlichen) periodischen Funktion deklariert werden, dann besteht die FT dieser periodischen Funktion aus den N Werten, die sich durch Auswertung von (5.88) ergeben. Die Schreibweise $\tilde{H}(n/NT)$ soll kennzeichnen, daß die DFT in der Regel immer nur eine Approximation der kontinuierlichen FT darstellt. In der Praxis läßt man zur Vereinfachung der Schreibarbeit diese Kennzeichnung weg und schreibt

$$H\left(\frac{n}{NT}\right) = \sum_{k=0}^{N-1} h(kT) \cdot \mathrm{e}^{-\mathrm{j}2\pi nk/N}, \ \text{mit} \quad n = 0, \ 1, \ldots, \ N-1. \tag{5.92}$$

5.6.1.1 Inverse diskrete Fouriertransformation

Die Beziehung für die inverse diskrete Fouriertransformation (IDFT) lautet:

$$h(kT) = \frac{1}{N} \sum_{n=0}^{N-1} H\left(\frac{n}{NT}\right) \cdot \mathrm{e}^{\mathrm{j}2\pi nk/N}, \quad \text{mit} \quad k = 0, \ 1, \ \ldots, \ N-1 \tag{5.93}$$

Um zu zeigen, daß (5.92) und (5.93) ein diskretes Fouriertransformationspaar bilden, wird (5.93) in (5.92) eingesetzt:

$$H\left(\frac{n}{NT}\right)=\sum_{k=0}^{N-1}\left[\frac{1}{N}\sum_{r=0}^{N-1}H\left(\frac{r}{NT}\right)\cdot \mathrm{e}^{\mathrm{j}2\pi rk/N}\right]\cdot \mathrm{e}^{-\mathrm{j}2\pi nk/N}=$$

$$=\frac{1}{N}\sum_{r=0}^{N-1}H\left(\frac{r}{NT}\right)\cdot\left[\underbrace{\sum_{k=0}^{N-1}\mathrm{e}^{\mathrm{j}2\pi rk/N}\cdot \mathrm{e}^{-\mathrm{j}2\pi nk/N}}_{\begin{array}{c}=N\ \text{für}\ r=n\\ =0\ \ \text{für}\ \ r\neq n\end{array}}\right]=H\left(\frac{n}{NT}\right) \tag{5.94}$$

Bei der Auswertung der eckigen Klammer in (5.94) wurde die bekannte Orthogonalitätsbeziehung verwendet.

Die IDFT-Beziehung (5.94) weist in gleicher Weise wie die DFT nach (5.92) eine Periodizität auf. Die Periode ist durch N Werte von $h(kT)$ gegeben. Diese Eigenschaft folgt aus der Periodizität von $\mathrm{e}^{\mathrm{j}2\pi \mathrm{n}/\mathrm{N}}$. Damit ist $h(kT)$ zwar für alle ganzzahligen Werte von k, (k=0, ±1, ±2, ±3, ...) wohldefiniert, unterliegt dabei aber der Identitätsbeziehung bezüglich der Periode N, d.h.

$$h\left(kT\right)=h\left[\left(rN+k\right)T\right],\ \text{mit}\ r=0,\ \ \pm 1,\ \ \pm 2,...$$

Das DFT-Paar lautet also zusammenfassend:

$$h\left(kT\right)=\frac{1}{N}\sum_{n=0}^{N-1}H\left(\frac{n}{NT}\right)\cdot \mathrm{e}^{\mathrm{j}2\pi nk/N}\ \circ\!\!-\!\!\bullet\ H\left(\frac{n}{NT}\right)=\sum_{k=0}^{N-1}h\left(kT\right)\cdot \mathrm{e}^{-\mathrm{j}2\pi nk/N} \tag{5.95}$$

Es ist wichtig, sich darüber im Klaren zu sein, daß dieses Transformationspaar stets eine Periodizität sowohl der Zeitfunktion als auch der Frequenzfunktion verlangt, d.h.

$$\begin{aligned} &H\left(\frac{n}{NT}\right)=H\left[\frac{\left(rN+n\right)}{NT}\right],\ \ \text{mit}\ \ r=0,\ \ \pm 1,\ \ \pm 2,...\\ &h\left(kT\right)=h\left[\left(rN+k\right)T\right],\qquad \text{mit}\ \ r=\ 0,\ \ \pm 1,\ \ \pm 2,\ ... \end{aligned} \tag{5.96}$$

Zusammengefaßt lauten die Bedingungen für die Identität von DFT und kontinuierlicher FT:

1.) die Zeitfunktion $h(t)$ muß periodisch sein

2.) $h(t)$ muß bandbegrenzt sein

3.) die Abtastfrequenz f_A muß mindestens doppelt so hoch sein wie die höchste spektrale Komponente von $h(t)$

4.) die Fensterfunktion $w(t)$ muß sich exakt über eine Periode von $h(t)$ oder einem Vielfachen davon erstrecken

Nur in diesem Fall gilt (5.95) ohne Einschränkung.

5.6.2 Fehlerbetrachtungen beim Übergang von der kontinuierlichen FT zur DFT

Die diskrete Fouriertransformation ist deswegen von großem Interesse, weil sie eine zur digitalen Verarbeitung geeignete Approximation der kontinuierlichen FT darstellt. Der Gütegrad dieser Approximation hängt erheblich von dem zu transformierenden Signal ab. Im folgenden wird exemplarisch für einige Signalklassen die Übereinstimmung von DFT und kontinuierlicher FT untersucht. Dabei wird offensichtlich, daß die Unterschiede zwischen den beiden Transformationen letztlich auf die für die DFT notwendigen Schritte ***Abtastung*** und ***Zeitfensterung*** zurückzuführen sind, weil diese in der Regel nicht ohne Fehler durchführbar sind.

5.6.2.1 Bandbegrenzte periodische Signale mit „Fensterung" über eine Periode

Man betrachte die Funktion $h(t)$ und ihre FT unter (a) in Bild 5.7. $h(t)$ soll abgetastet werden, das Abtastsignal soll auf N Abtastwerte zeitbegrenzt werden, und das Ergebnis ist schließlich der DFT zu unterziehen. Die einzelnen Schritte sind unter (b-g) in Bild 5.7 graphisch veranschaulicht. $h(t)$ wird durch Multiplikation mit der Abtastfunktion unter (b) abgetastet. (c) zeigt das Abtastsignal und seine Fouriertransformierte. Man beachte, daß in diesem Beispiel keine Überlappung (Aliasing) entsteht, und daß das Spektrum wegen der Zeitbereichsskalierung mit dem Faktor $1/T$ zu multiplizieren ist. Deswegen haben die Diracimpulse der FT (unter c, links) statt dem ursprünglichen Gewicht $A/2$ nun das Gewicht $A/2T$. Das Abtastsignal wird durch Multiplikation mit der Rechteckfunktion unter (d) zeitbegrenzt. Unter (e) erhält man das zeitbegrenzte Abtastsignal. Die Fensterfunktion ist wieder derart gewählt, daß die nach der Begrenzung verbleibenden N Abtastwerte genau eine Periode des Originalsignals $h(t)$ umfassen.

Die FT des zeitbegrenzten Abtastsignals unter (e) erhält man durch die Faltung der Diracimpulse aus (c) mit der ***si***-Funktion aus (d), die die FT der Fensterfunktion repräsentiert. Bild 5.7 (e) zeigt rechts das Faltungsprodukt.

Im Vergleich zur ursprünglichen FT $H(f)$ ist das durch den zuletzt beschriebenen Faltungsschritt entstandene Spektrum (e) stark verzerrt. Wenn man dieses jedoch mit der unter (f) dargestellten Funktion $\sum_{r=-\infty}^{\infty} \delta\left(f - r/T_0\right)$ abtastet, wird die Verzerrung eliminiert. Dies ist deswegen gewährleistet, weil der Abstand der Diracimpulse der Frequenzbereichs-Abtastfunktion genau $1/T_0$ beträgt und somit bei allen Frequenzen außer bei den $\pm 1/T_0$ eine Nullstelle vorliegt. Die Stellen $\pm 1/T_0$ entsprechen exakt den Auftrittsorten der Diracimpulse der ursprünglichen FT $H(f)$ aus Bild 5.7 (c).

Zur Verdeutlichung der Zusammenhänge ist der soeben beschriebene Sachverhalt in Bild 5.9 (oben) noch mal vergrößert dargestellt. Die senkrechten Gitterlinien markieren die Abtastpunkte im Frequenzbereich, wobei nur die beiden zentralen von Null verschiedene Werte liefern. Somit liegt eine völlig fehlerfreie DFT vor, die bis auf einen Skalierfaktor ($1/T$) genau das gleiche Ergebnis wie eine kontinuierliche FT liefert. Wenn die FT mittels DFT ermittelt wurde soll, muß demnach das Ergebnis noch mit T multipliziert werden, damit man numerische Übereinstimmung erhält, d.h. die Transformationsgleichung lautet

$$H\left(\frac{n}{NT}\right) = T\sum_{k=0}^{N-1} h(kT)\cdot \mathrm{e}^{-\mathrm{j}2\pi nk/N} \,. \tag{5.97}$$

Das behandelte Beispiel repräsentiert die ***einzige*** Signalklasse, für die die Ergebnisse der DFT und der kontinuierlichen FT bis auf eine Konstante identisch sind. Denn nur hier sind die Voraussetzungen erfüllt:

- $h(t)$ ist periodisch und bandbegrenzt
- die Fensterung mit $w(t)$ erfaßt exakt eine Periode von $h(t)$ oder Vielfaches davon

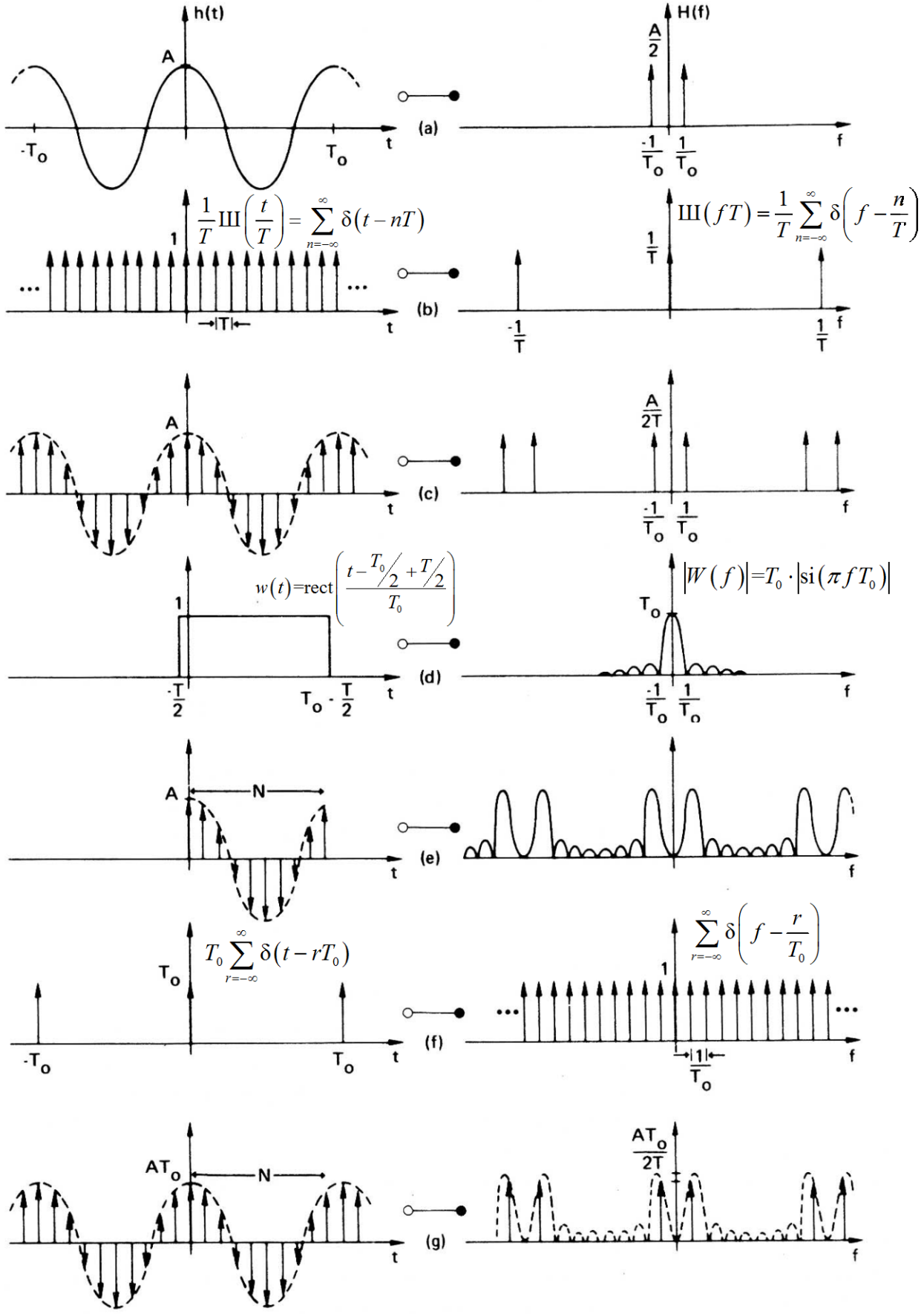


Bild 5.7: Fehlerfreie Ausführung der DFT durch richtige Fensterung

5.6.2.2 Bandbegrenzte periodische Funktionen, Beobachtungszeit ungleich einer Periode

Wenn eine periodische und bandbegrenzte Funktion abgetastet und derart zeitbegrenzt wird, daß sie nach der Zeitbegrenzung nicht aus einem ganzzahligen Vielfachen der Periode der ursprünglichen Funktion besteht, dann können sich die kontinuierliche und die diskrete FT ***erheblich*** unterscheiden. Um diesen Effekt zu zeigen, werden im folgenden Bild 5.8 die Grafiken unter (a-g) analysiert. Dieses Beispiel unterscheidet sich vom vorigen aus Bild 5.7 lediglich in der Frequenz des sinusförmigen Signals $h(t)$. Wie vorher, wird $h(t)$ abgetastet (c) und zeitbegrenzt (e). Man beachte, daß das zeitbegrenzte Abtastsignal jetzt aber keinem ganzzahligen Vielfachen einer Periode von $h(t)$ entspricht. Somit resultiert aus der Zeitbereichsfaltung der Funktionen aus (e) und (f) in Bild 5.8 die unter (g) links dargestellte ***periodische*** Funktion. Obwohl diese Funktion periodisch ist, unterscheidet sie sich erheblich von $h(t)$. Es kann somit nicht erwartet werden, daß die FTs der beiden Funktionen unter (a) und (g) äquivalent sind. Für einen genaueren Einblick ist es sinnvoll, die Zusammenhänge im Frequenzbereich näher zu untersuchen.

Die FT des zeitbegrenzten Abtastsignals aus Bild 5.8 (e) wurde durch die Faltung der Dirac-Impulsfolge im Frequenzbereich unter (c) (links) mit der si-Funktion aus (d) gewonnen.

Zur Verdeutlichung ist der Sachverhalt in Bild 5.9 (unten) vergrößert dargestellt. Die senkrechten Gitterlinien markieren wieder die Abtastpunkte im Frequenzbereich, wobei aber jetzt mehrere - und nicht nur die beiden zentralen - von Null verschiedene Werte liefern. Somit liegt ganz offenbar keine fehlerfreie DFT mehr vor. Es erscheinen zahlreiche Spektrallinien, die das ursprüngliche Signal $h(t)$ definitiv nicht aufweist. Sogar bei der Frequenz Null, die den Mittelwert des zeitbegrenzten Signals repräsentiert tritt ein Diracimpuls auf, d.h. es wird eine Gleichanteil vorgetäuscht.

Diese Diskrepanz zwischen der kontinuierlichen und der diskreten FT tritt in der Praxis wahrscheinlich am häufigsten auf und wird von vielen Anwendern der DFT am wenigsten verstanden. Die Zeitbegrenzung auf ein nicht ganzzahliges Vielfaches der Periode führt zu einer periodischen Funktion mit scharfen Diskontinuitäten. Intuitiv erwartet man aufgrund der Sprünge im Zeitbereich die Entstehung zusätzlicher Spektralkomponenten im Frequenzbereich. Eine Zeitbegrenzung führt im Frequenzbereich stets zu einer Faltung mit einer si-Funktion. So entsteht eine Art Frequenzkontinuum mit einem Maximum und einer Reihe weiterer glockenförmiger Frequenzanteile, die als Nebenzipfel (engl.: side lobes) bezeichnet werden. Dieses Phänomen ist auch unter dem Namen Leckeffekt (engl.: leakage) bekannt und ist eine Eigenheit der DFT. Ursache ist die Notwendigkeit zur Zeitfensterung. Methoden zur Abschwächung des Leckeffekts werden im weiteren behandelt.

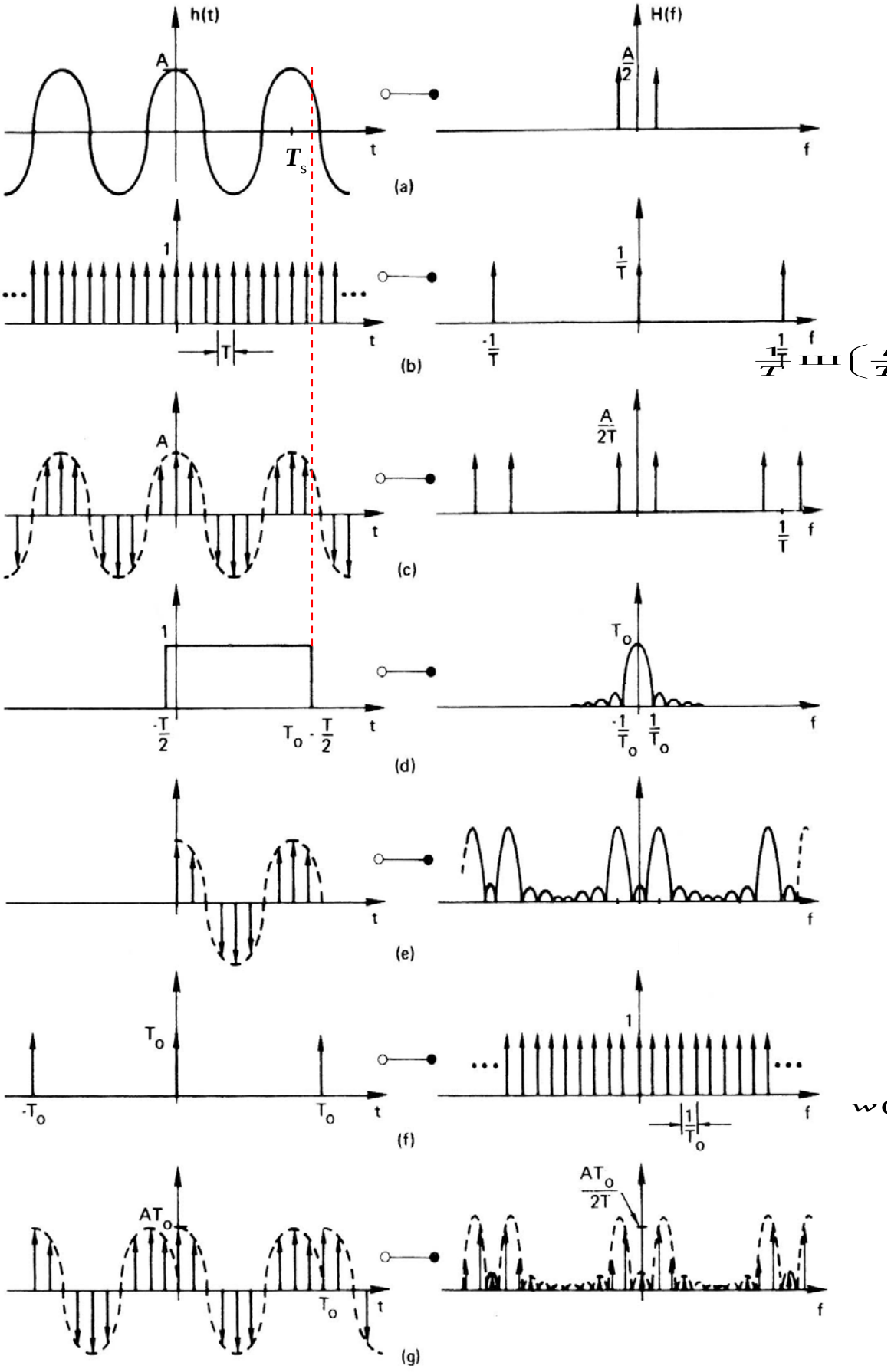


Bild 5.8: Fehlerhafte Ausführung der DFT durch falsche Fensterung

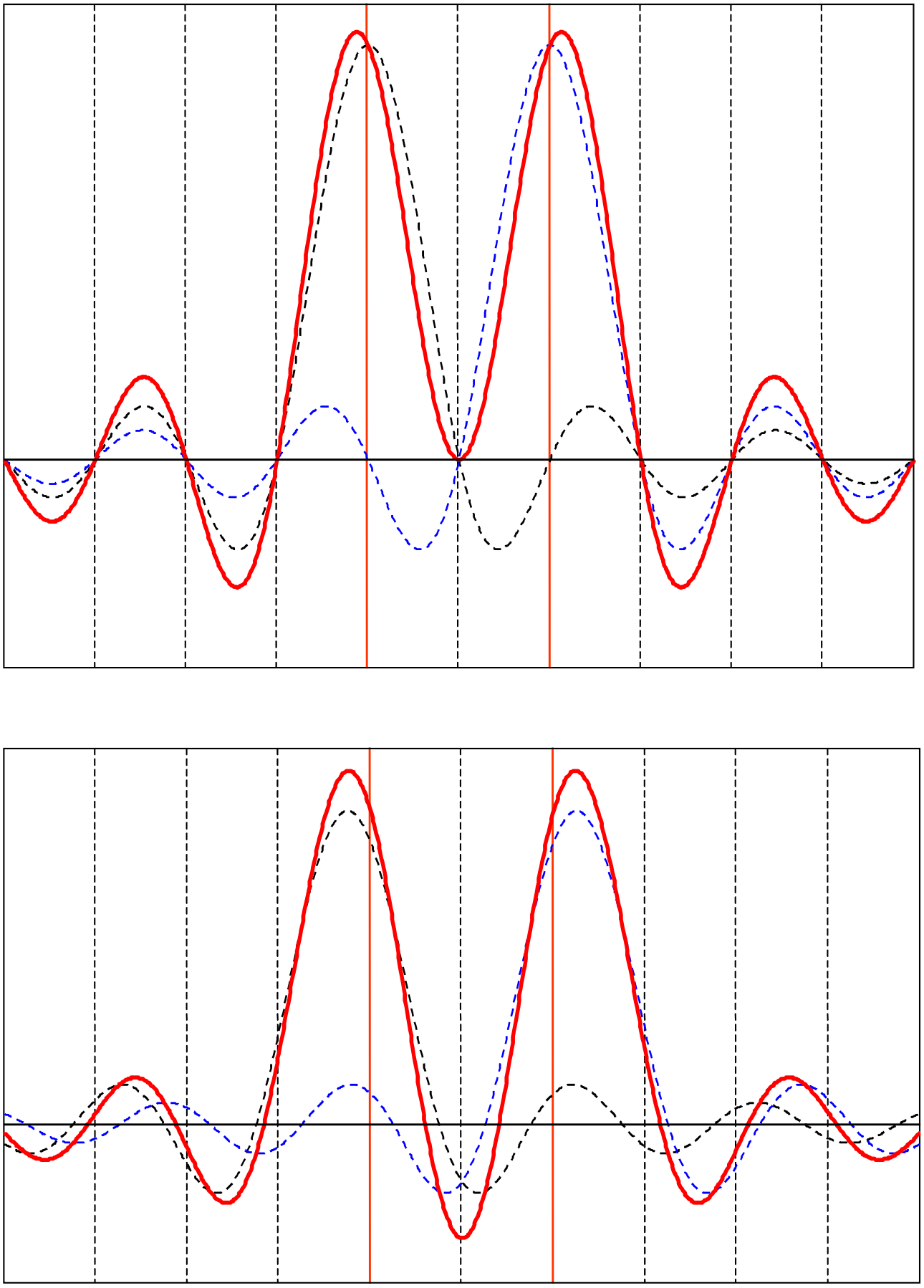


Bild 5.9: Details der Spektren aus Bild 5.7 und Bild 5.8 bei korrekter (oben) und fehlerhafter Fensterung (unten).

5.6.2.3 Beliebige Signale

Die in der Praxis weitaus wichtigste Signalklasse ist weder bandbegrenzt noch zeitbegrenzt. Im folgenden Bild 5.10 wird unter (a-g) ein Beispiel dazu behandelt. Die Abtastung führt, wie aus Bild 5.10 (c) hervorgeht, zu überlappenden Spektren. Zeitbegrenzung ruft – wie (e) zeigt - Welligkeiten im Frequenzbereich hervor. Die Frequenzbereichsabtastung ergibt das unter (g) dargestellte Fouriertransformationspaar. Die Zeitfunktion dieses Transformationspaares ist eine periodische Funktion, bei der jede Periode sich aus den durch Abtastung und Zeitbegrenzung entstandenen N Werten der ursprünglichen Funktion zusammensetzt. Die Frequenzfunktion ist ebenfalls periodisch, wobei jede Periode aus N Werten besteht, die sich von den entsprechenden Werten des ursprünglichen Spektrums durch die auf Abtastung und Zeitbegrenzung zurückzuführenden Fehler unterscheiden. Der Aliasingfehler läßt sich durch Verkleinerung des Abtastintervalls verringern. Verfahren zur Verringerung der durch Zeitbegrenzung entstehenden Fehler werden im weiteren anhand spezieller Fensterfunktionen diskutiert.

Aus den hier angestellten Betrachtungen geht hervor, daß es viele Anwendungsfälle gibt, für die man mit Hilfe der DFT nur mit der nötigen Vorsicht brauchbare Resultate erzielen kann. Dabei darf nie vergessen werden, daß die DFT grundsätzliche eine Periodizität sowohl im Zeitbereich als auch im Frequenzbereich erfordert.

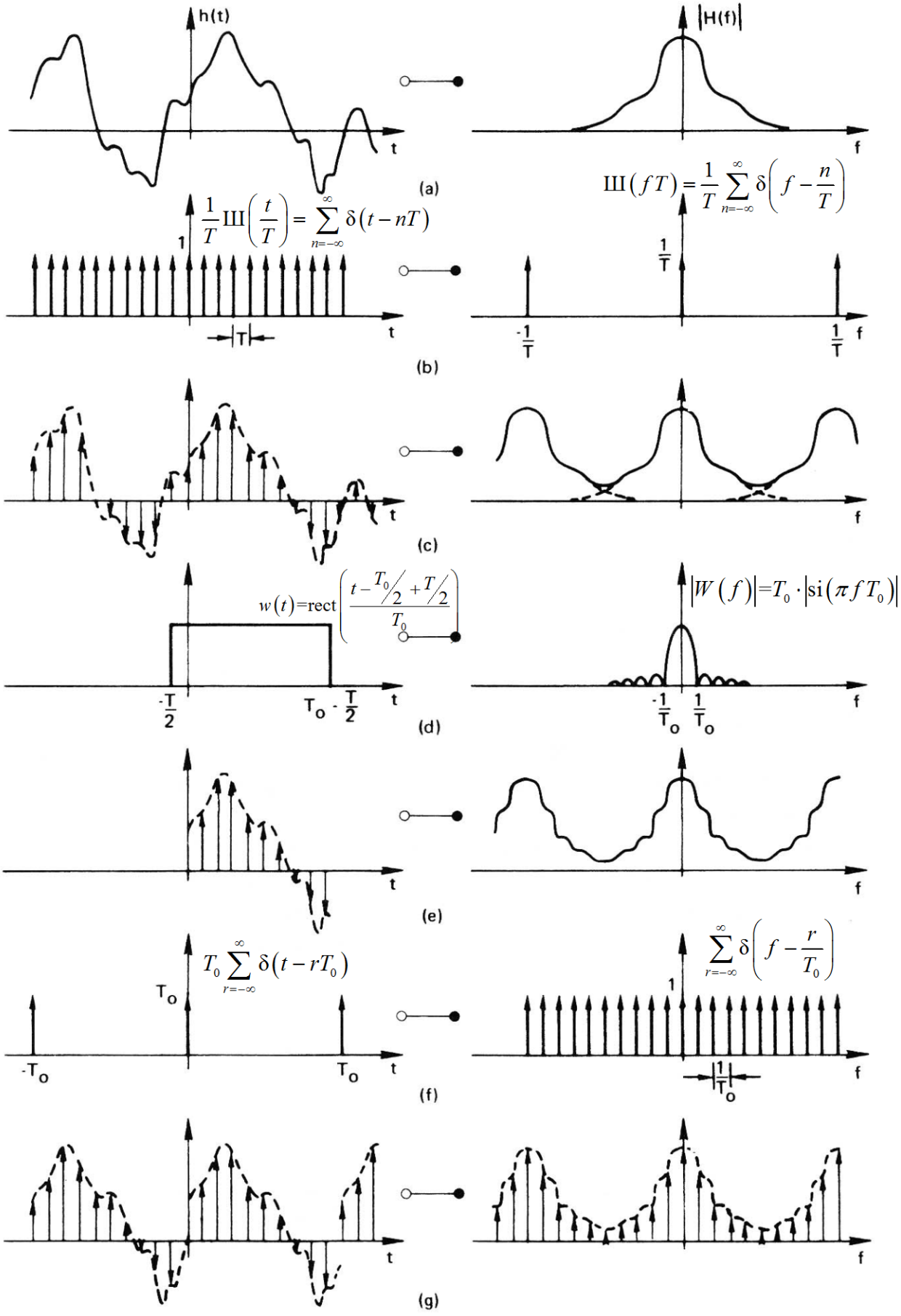


Bild 5.10: DFT eines allgemeinen Signals, das weder periodisch noch bandbegrenzt ist

5.7 Anwendungen der DFT

Der Zusammenhang zwischen der DFT und der kontinuierlichen FT wurde ausführlich behandelt. Im weiteren wird anhand einiger Beispiele die praktische Anwendung der DFT untersucht. Eine korrekte Interpretation der Ergebnisse wird sich dabei als sehr wichtig erweisen.

Als einführendes Beispiel wird die nicht zeitbegrenzte Funktion e^{-t} betrachtet. Das Spektrum dieser Funktion ist mit Hilfe der DFT näherungsweise zu berechnen.

Der erste Schritt ist die Wahl der Anzahl N der Abtastwerte und die Länge T des Abtastintervalls.

In Bild 5.11 sind die Abtastwerte von e^{-t} für N=32 und T=0,25 als senkrechte Linien eingetragen. Man beachte, daß der Abtastwert bei $t = 0$ so gewählt ist, daß der Wert an der Sprungstelle gleich dem Mittelwert der Funktionswerte auf beiden Seiten der Sprungstelle ist. Damit wird das Unstetigkeitsproblem abgemildert. Würde man den Wert 1 bei t=0 beibehalten, dann würden alle Fourierkoeffizienten zu groß bestimmt.

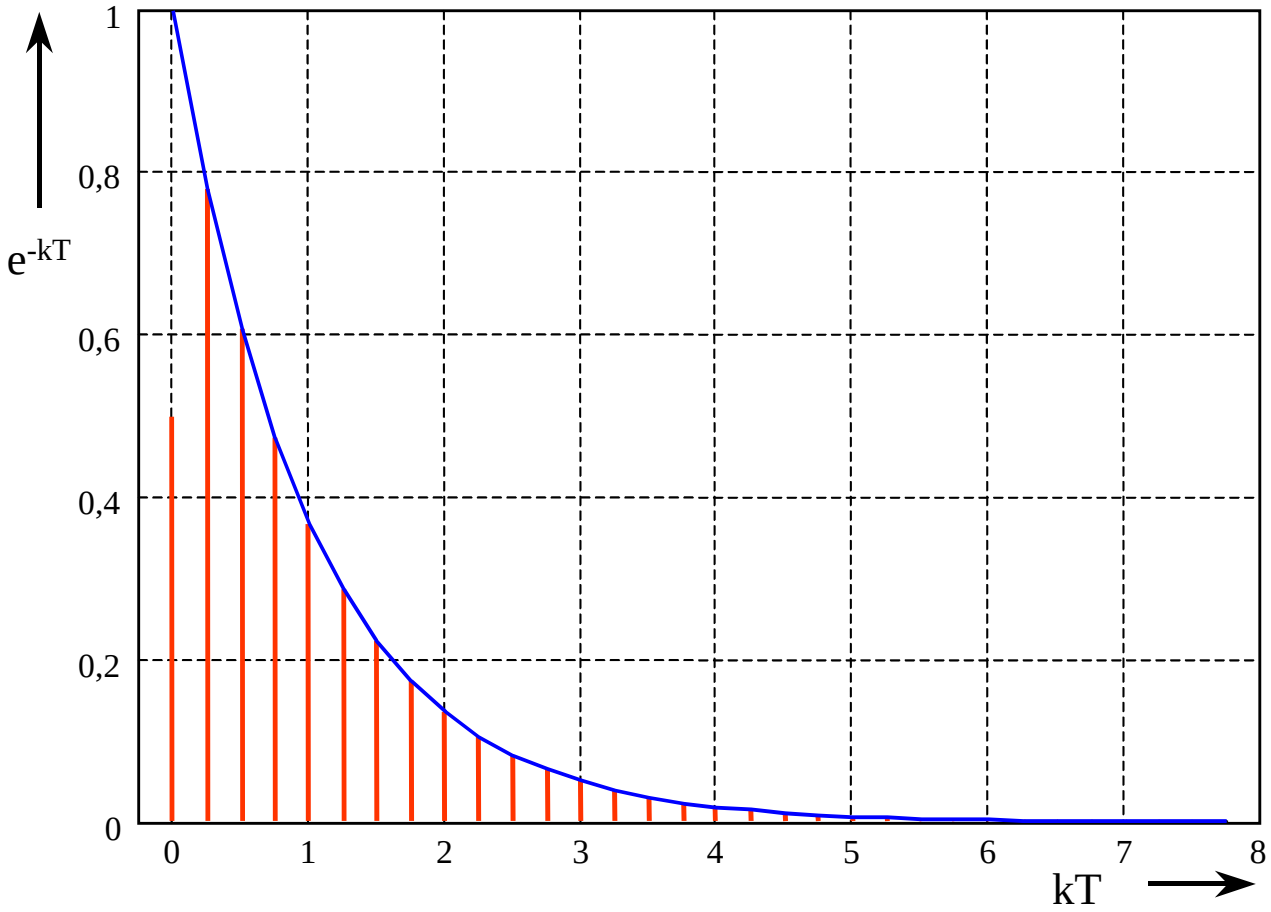


Bild 5.11: Abtastung der Zeitfunktion mit Modifikation der Sprungstelle bei 0

Im nächsten Schritt wird die DFT unter Verwendung von

$$H\left(\frac{n}{NT}\right) = T \cdot \sum_{k=0}^{N-1} \left[e^{-kT} \right] \cdot e^{-j2\pi nk/N}, \quad \text{mit } n = 0,1,2...N-1 \qquad (5.98)$$

berechnet. Man beachte daß der Faktor T hinzugefügt werden muß, um numerisch die Gleichheit von DFT und kontinuierliche FT herzustellen. In Bild 5.12 sieht man den Realteil $R(n)$ der kontinuierlichen FT im Vergleich zum Ergebnis der numerischen Auswertung von (5.98).

Man beachte, daß der Realteil der DFT bezüglich $n=N/2$ symmetrisch ist. Dies folgt daraus, daß erstens der Realteil gerade ist, und daß zweitens die Werte der DFT für $n > N/2$ mit den Werten für negative Frequenzen identisch sind.

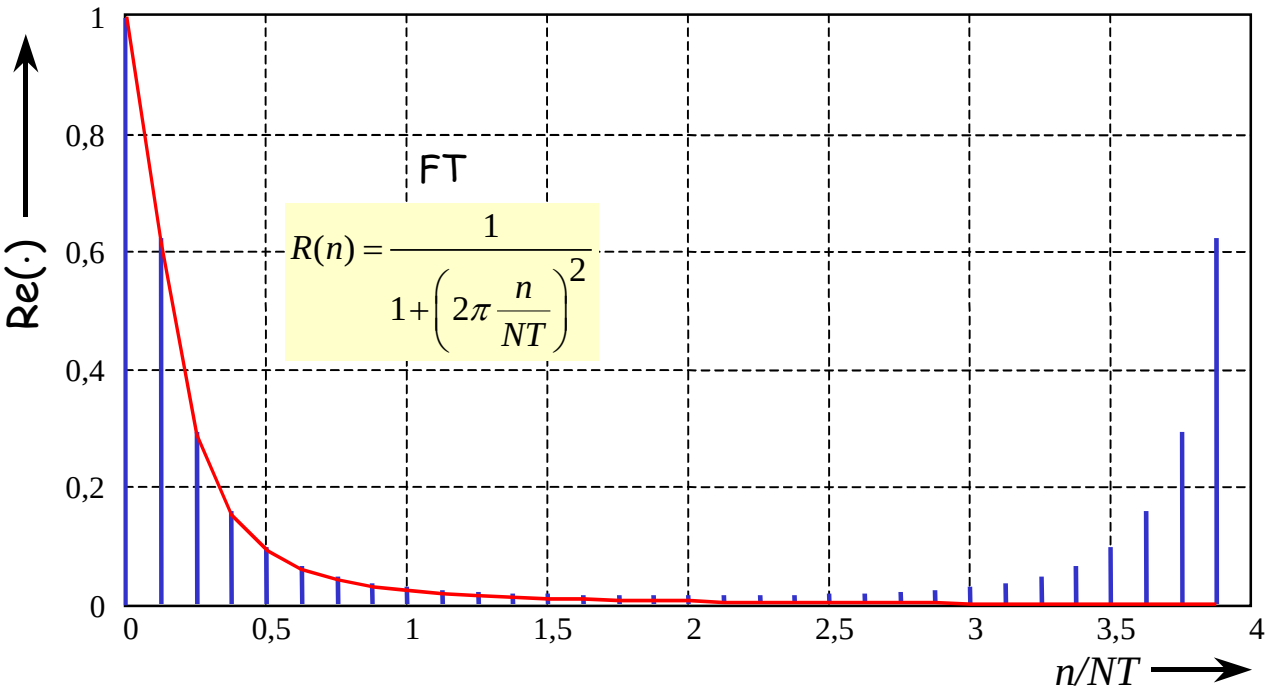


Bild 5.12: Realteilvergleich von DFT und kontinuierlicher FT

In Bild 5.13 ist der Imaginärteil $I(n)$ der kontinuierlichen FT dem der DFT gegenübergestellt. Die DFT zeigt bei höheren Frequenzen relativ große Abweichungen von der FT. Durch eine Verkleinerung des Abtastintervalls T und einer Erhöhung von N läßt sich der Approximationsfehler verringern.

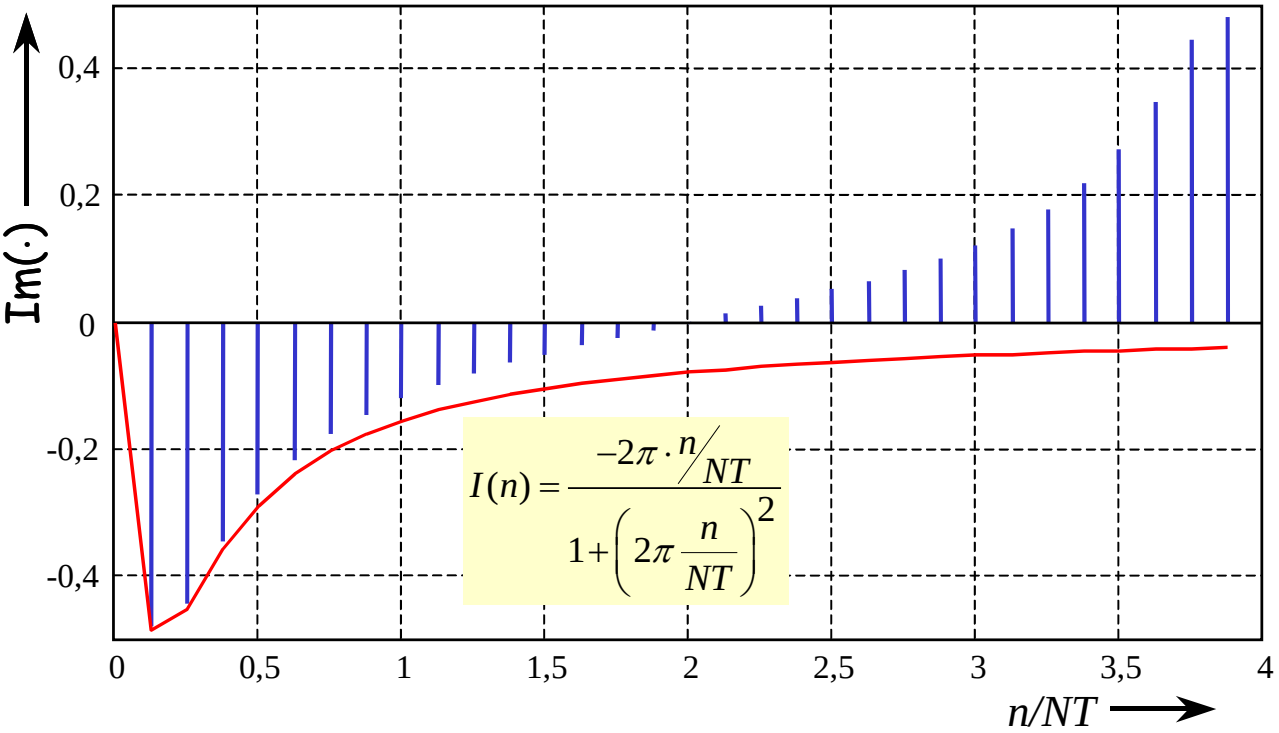


Bild 5.13: Imaginärteilvergleich von DFT und kontinuierlicher FT

Man beachte, daß der Imaginärteil bezüglich $n=N/2$ ungerade ist. Auch hier sind die Ergebnisse für $n > N/2$ wieder als Ergebnisse für negative Frequenzen zu interpretieren.

5.7.1.1 Approximation der inversen Fouriertransformation

Ausgehend von den Ergebnissen des vorangegangenen Abschnitts soll jetzt die zugehörige Zeitfunktion unter Anwendung der IDFT

$$h(kT) = \frac{1}{NT} \cdot \sum_{n=0}^{N-1} \left[R\left(\frac{n}{NT}\right) + \mathrm{j} \cdot I\left(\frac{n}{NT}\right) \right] \cdot \mathrm{e}^{\mathrm{j}2\pi nk/N}, \text{ mit } k = 0,1,...,N-1 \qquad (5.99)$$

zu berechnen, wobei $1/NT=\Delta f$ das Abtastintervall im Frequenzbereich ist. Mit N=32 und T=0,25 – wie oben - erhält man Δf=1/8.

$$h(kT) = \frac{1}{NT} \cdot \sum_{n=0}^{N-1} \left[\mathrm{Re}(c(n)) + \mathrm{j}\,\mathrm{Im}(c(n)) \right] \cdot \mathrm{e}^{\mathrm{j}2\pi nk/N}$$

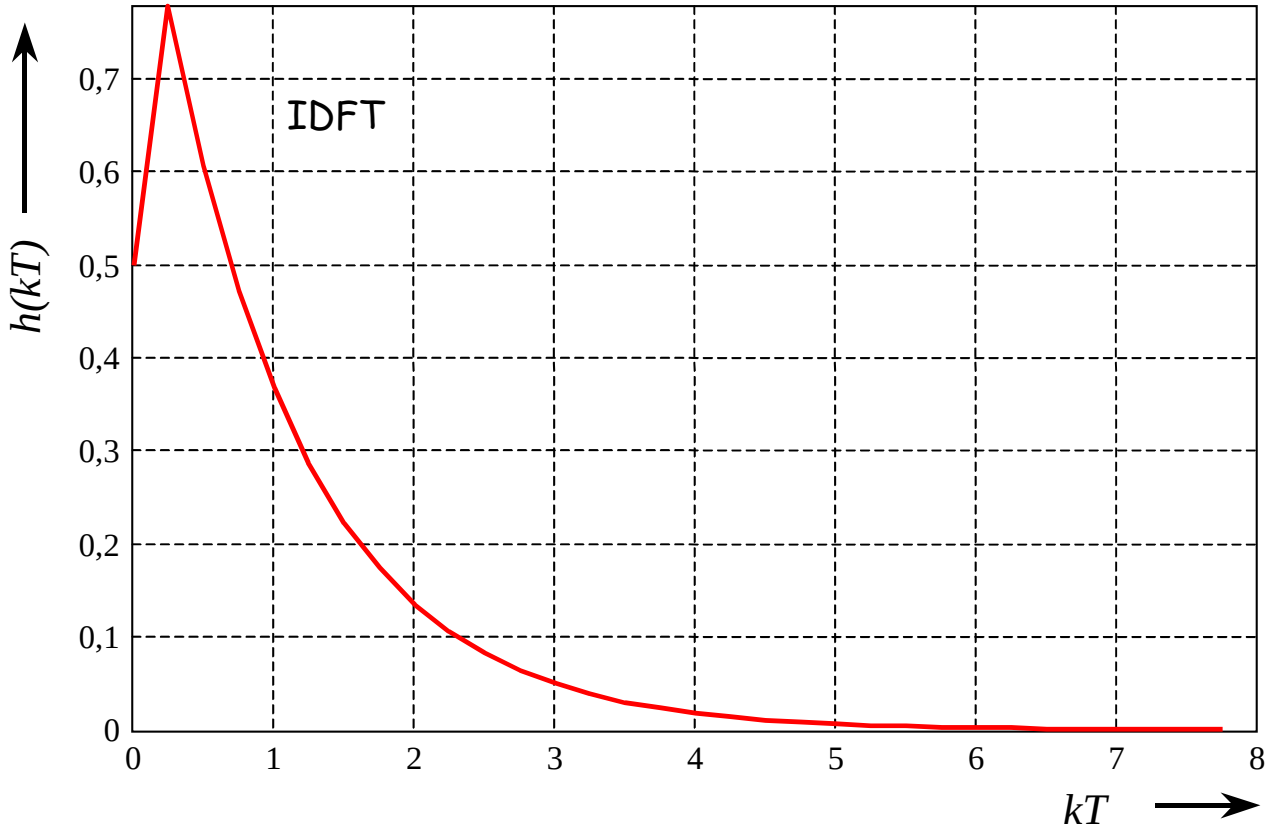


Bild 5.14: IDFT zur Rückgewinnung des Zeitsignals

Die Anwendung von (5.99) auf die diskreten Funktionen aus Bild 5.12 und Bild 5.13 liefert die IDFT. Sie ist prinzipiell eine komplexe Funktion, deren Imaginärteil jedoch annähernd gleich Null ist, so daß ihr Realteil den in Bild 5.14 dargestellten Verlauf hat. Man beachte, daß das Ergebnis bei k=0 gleich dem Mittelwert 0,5 des Sprunges ist. Wie man sieht, wird der gesamte übrige Verlauf der ursprünglichen Zeitfunktion recht gut approximiert. Wenn nötig lassen sich weitere Verbesserungen durch Verkleinerung von Δf und eine Erhöhung von N erreichen.

Durch Anwendung der alternativen Inversionsbeziehung kann das gleiche Ergebnis erzielt werden. Um diese Beziehung anwenden zu können, ist zunächst die konjugiert–komplexe Frequenzfunktion zu bilden. Dazu wird die diskrete imaginäre Funktion von Bild 5.13 mit –1 multipliziert. Da die resultierende Zeitfunktion reell ist, erübrigt sich die Bildung des konjugiert-komplexen Klammerin-

haltes. Deswegen ist nur der folgende Ausdruck zu berechnen, um die Zeitfunktion von Bild 5.14 zu erhalten.

$$h(kT) = \frac{1}{NT} \cdot \sum_{n=0}^{N-1} \left[R\left(\frac{n}{NT}\right) - \mathrm{j} \cdot I\left(\frac{n}{NT}\right) \right] \cdot \mathrm{e}^{-\mathrm{j}2\pi nk/N}, \text{ mit } k = 0,1,...,N-1 \qquad (5.100)$$

5.7.1.2 Maßnahmen zu Verringerung des Leckeffekts

Wie aus Bild 5.8 hervorgeht, treten bei feh1erhafter „Fensterung“ bei der DFT zusätzliche spektrale Komponenten auf, die im Ursprungssignal definitiv nicht enthalten waren. Diese „Artefakte“ werden Leckkomponenten genannt und sind auf die Nebenzipfel der si-Funktion zurückzuführen, die wegen der falschen Fensterbreite an den Abtastpunkten nicht mehr verschwinden. Der Leckeffekt läßt sich abschwächen, wenn man eine Fensterfunktion verwendet, deren Nebenzipfel deutlich niedriger sind als die der si-Funktion. Der Leckeffekt wird um so besser unterdrückt, je kleiner die Nebenzipfel sind. Es gibt zahlreiche Varianten von Fensterfunktionen, die alle das Ziel haben, möglichst niedrige Nebenzipfel aufzuweisen.

Eine häufig angewandte Fensterfunktion, die hier exemplarisch betrachte wird, ist das „Hanning“–Fenster, das durch die Gleichung

$$w(t) = \mathrm{rect}\left(\frac{t - \frac{T_0}{2}}{T_0} \right) \left[\frac{1}{2} - \frac{1}{2} \cdot \cos\frac{2\pi t}{T_0} \right] \qquad (5.101)$$

definiert ist, mit $T_0 = 1/f_0$ als Fensterbreite. Um einen Eindruck der Höhe der Nebenzipfeldämpfung zu erhalten, wird das Spektrum $W(f)$ der Fensterfunktion dargestellt. Die FT von (5.101) ergibt

$$\begin{aligned}
W(f) &= T_0 \cdot \mathrm{si}(\pi f T_0) \cdot \mathrm{e}^{-\mathrm{j}2\pi f \frac{T_0}{2}} * \left\{ \frac{1}{2} \cdot \delta(t) - \frac{1}{4} \cdot \left[\delta(f - f_0) + \delta(f + f_0) \right] \right\} = \\
&= \frac{T_0}{2} \cdot \mathrm{si}(\pi f T_0) \cdot \mathrm{e}^{-\mathrm{j}\pi f T_0} - \frac{T_0}{4} \mathrm{si}\left(\pi T_0 (f - f_0)\right) \mathrm{e}^{-\mathrm{j}\pi (f - f_0) T_0} - \frac{T_0}{4} \mathrm{si}\left(\pi T_0 (f + f_0)\right) \mathrm{e}^{-\mathrm{j}\pi (f + f_0) T_0} = \\
&= \frac{T_0}{2} \cdot \left\{ \mathrm{si}(\pi f T_0) - \frac{1}{2} \mathrm{si}\left(\pi T_0 (f - f_0)\right) \underbrace{\mathrm{e}^{\mathrm{j}\pi f_0 T_0}}_{-1} - \frac{1}{2} \mathrm{si}\left(\pi T_0 (f + f_0)\right) \underbrace{\mathrm{e}^{-\mathrm{j}\pi f_0 T_0}}_{-1} \right\} \cdot \mathrm{e}^{-\mathrm{j}\pi f T_0} = \\
&= \frac{T_0}{2} \cdot \left\{ \mathrm{si}(\pi f T_0) + \frac{1}{2} \mathrm{si}\left(\pi T_0 (f - f_0)\right) + \frac{1}{2} \mathrm{si}\left(\pi T_0 (f + f_0)\right) \right\} \cdot \mathrm{e}^{-\mathrm{j}\pi f T_0}
\end{aligned} \qquad (5.102)$$

Für den Betrag der Fouriertransformierten des „Hanning“–Fensters erhält man somit

$$|W(f)| = \frac{T_0}{2} \cdot \mathrm{si}(\pi f T_0) + \frac{T_0}{4} \left[\mathrm{si}\left(\pi T_0 (f + f_0)\right) + \mathrm{si}\left(\pi T_0 (f - f_0)\right) \right]. \qquad (5.103)$$

$W(f)$ weist sehr kleine Nebenzipfel auf. Es gibt zahlreiche andere Fensterfunktionen mit ähnlichen Eigenschaften. Wegen seiner Einfachheit wird das ’Hanning’–Fenster in der Praxis gerne angewandt.

Generell werden durch alle ’guten’ Fensterfunktionen die Leckkomponenten aufgrund der niedrigen Nebenzipfel wesentlich reduziert. Der interessierende Frequenzbereich wird dabei allerdings immer beträchtlich verbreitert. Dieser Effekt einfach erklärbar, da die Zeitbegrenzung immer zu einer Faltung mit der FT der Fensterfunktion führt. Je mehr man die Leckkomponenten unterdrückt, desto breiter oder „verschmierter“ erscheinen die interessierenden Ergebnisse der DFT. Die Kunst beim

Entwurf von guten Fensterfunktionen ist, einen ’angepaßten’ Kompromiß für die jeweils zu untersuchenden Signale zu finden.

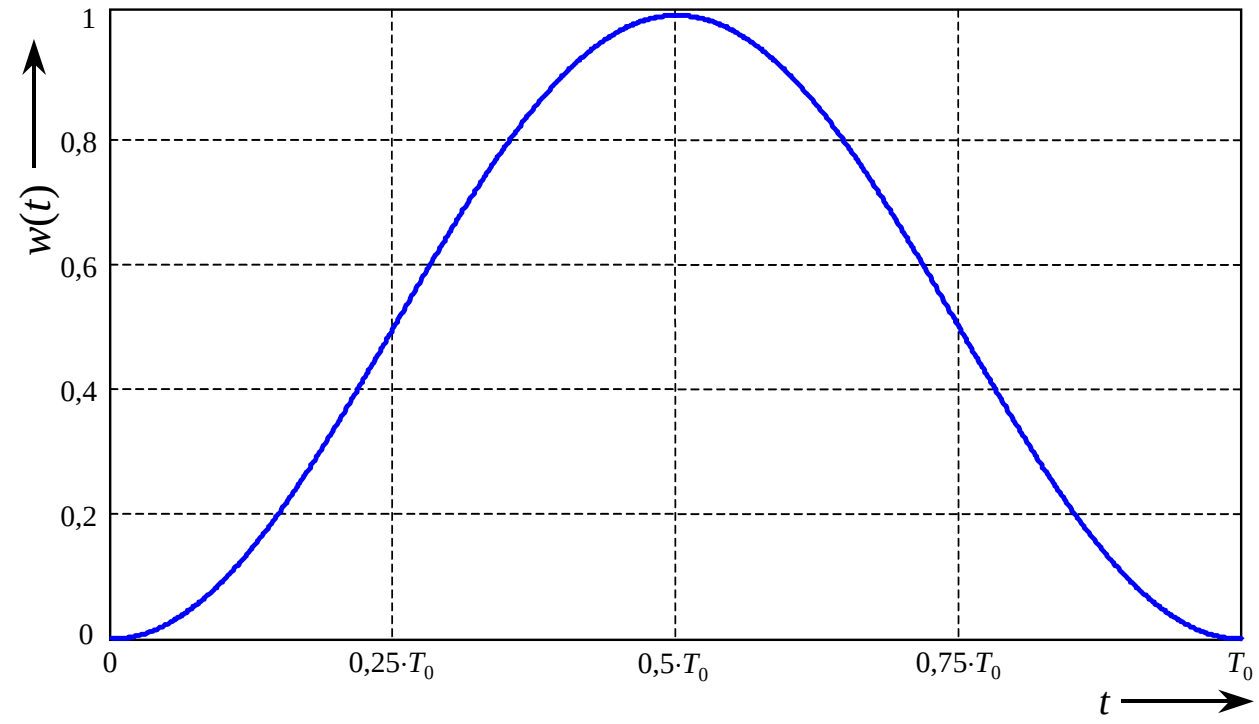


Bild 5.15: Zeitverlauf der Hanning–Fensterfunktion nach (5.101)

Wie man aus Bild 5.15 ersieht, „zwingt“ das Hanning–Fenster die Signale an seinen Rändern in relativ „sanfter“ Weise auf Null, d.h. es gibt keine abrupten Übergänge, so daß im Spektrum keine künstlich erzeugten hochfrequenten Komponenten zu erwarten sind. Dieser Aspekt wird bei Betrachtung der FT $|W(f)|$von $w(t)$ nach (5.103) deutlich.

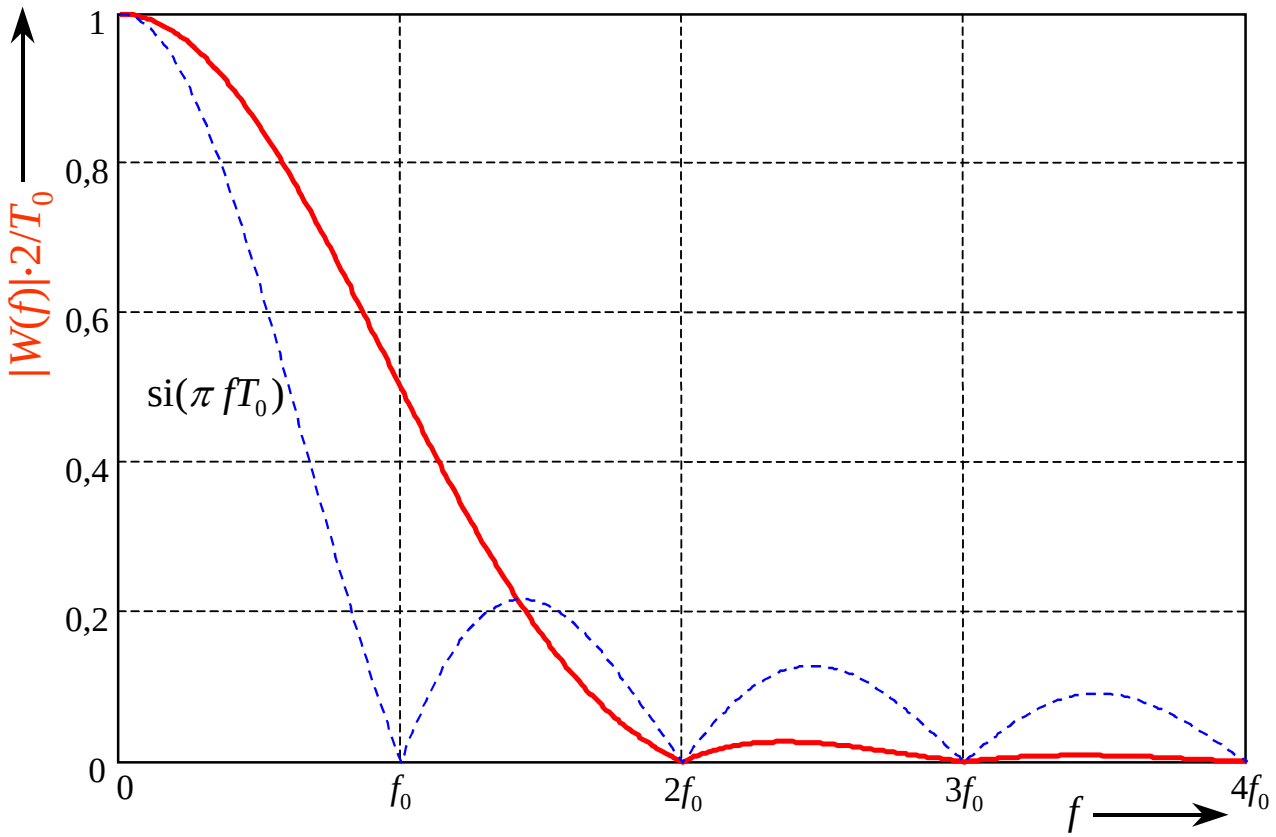


Bild 5.16: Normierte FT der Hanning–Fensterfunktion im Vergleich zur FT des Rechteckfensters

Bild 5.17 gibt einen Überblick über Form und Eigenschaften einiger technisch wichtiger Bewertungsfenster.

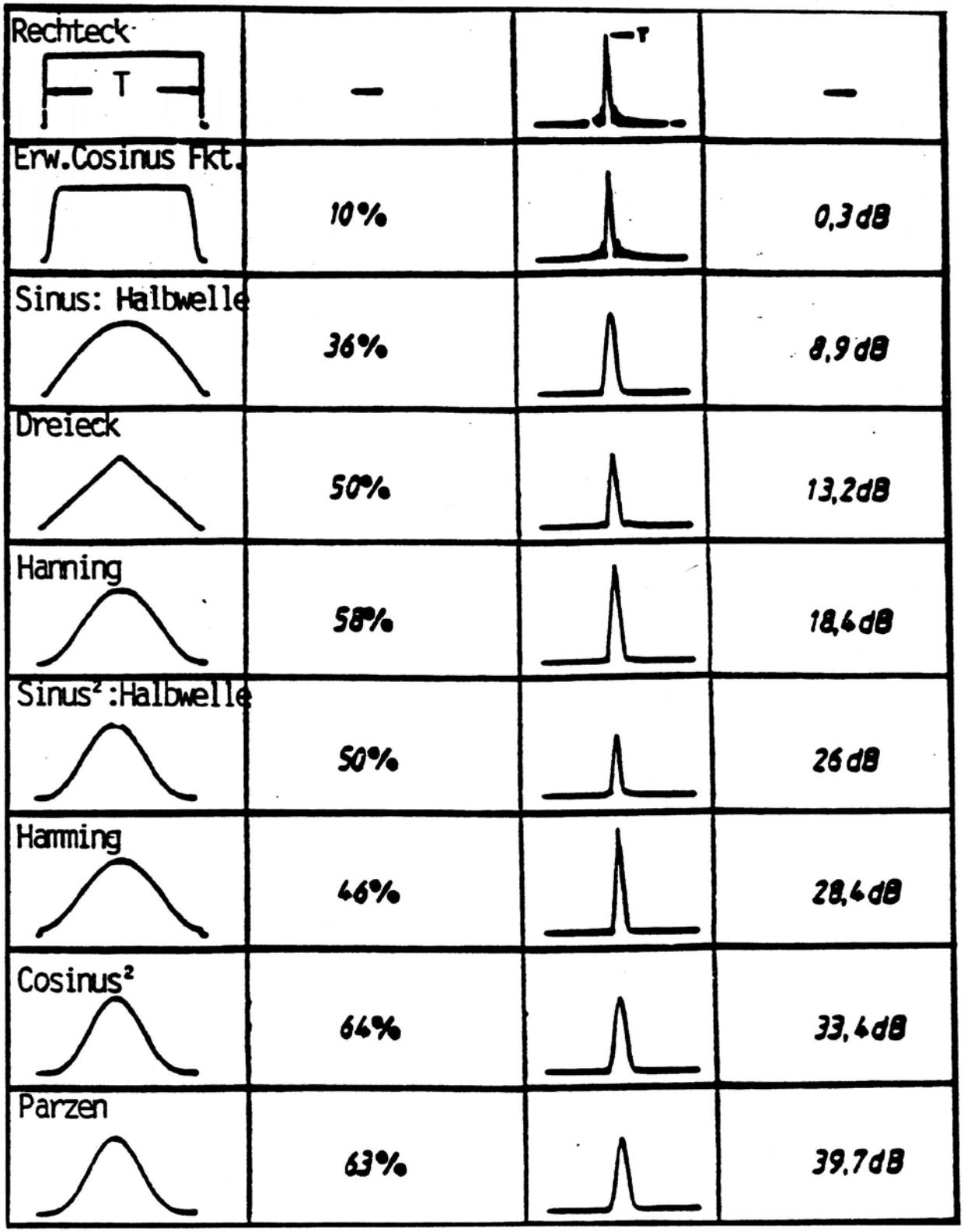


Bild 5.17: Verschiedene Fensterfunktionen für die DFT und ihre Haupteigenschaften

Die Vorteile des Einsatzes von Fensterfunktionen werden abschließend an einem einfachen Signalbeispiel verdeutlicht. Dabei wird zunächst ein Cosinussignal der Frequenz 1 Hz über exakt 2 Perioden mit N=64 Abtastwerten abgetastet. Mit dem Abtastraster von T=1/32s erhält man dann eine Frequenzauflösung von 0,5Hz – s. Bild 5.18.

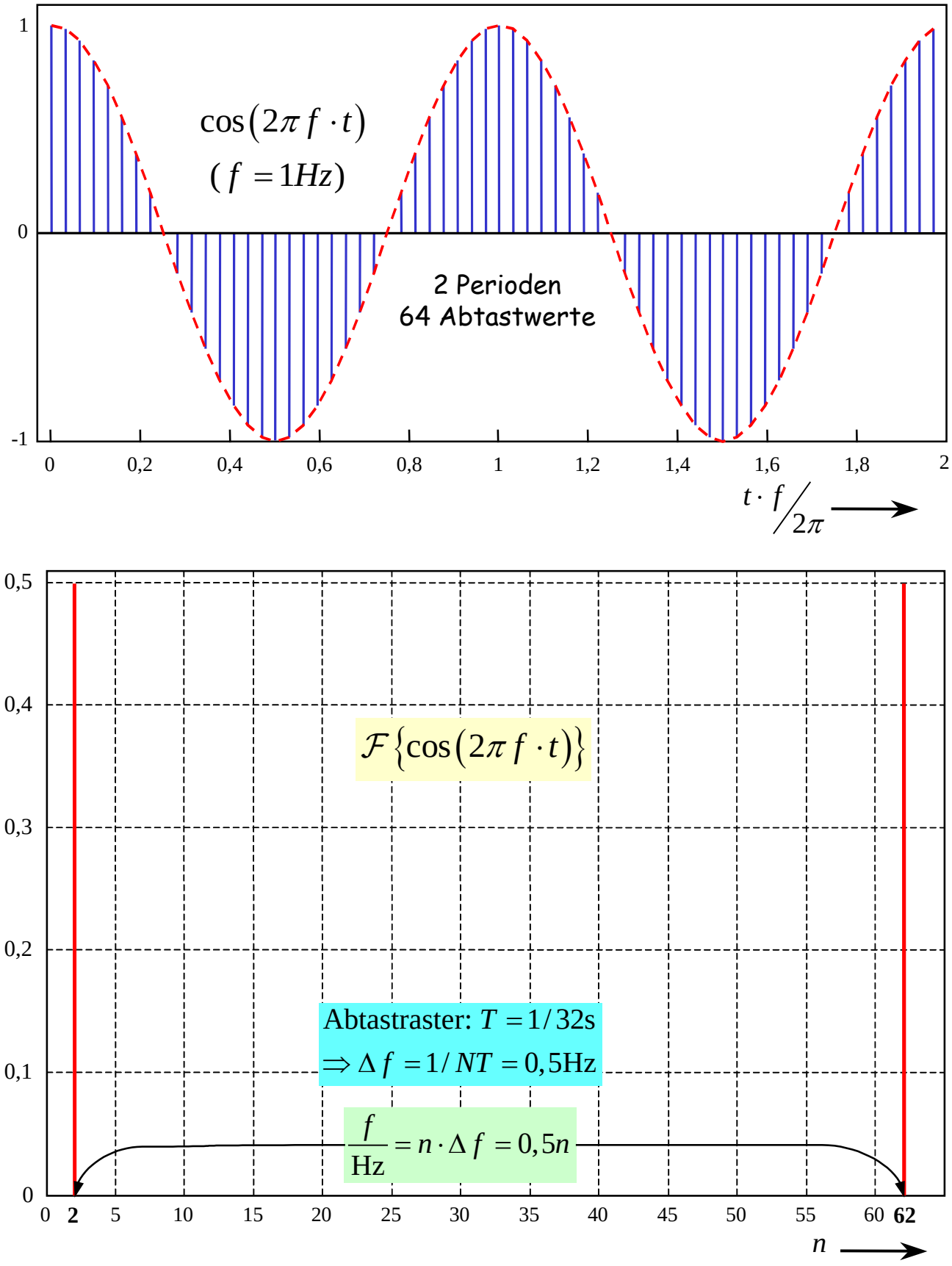


Bild 5.18: Korrekte Fensterung eines Cosinussignals liefert exakte Übereinstimmung von DFT und kontinuierlicher FT

Wie man sieht tauchen nur an den Stellen n_1=2 und n_2=N-n_1=62 Spektrallinien der Höhe 0,5 auf, was genau den bei analytischer Berechnung der kontinuierlichen FT erwarteten Werten bei +1Hz und –1Hz entspricht.

Diese idealen Gegebenheiten ändern sich drastisch, wenn die Frequenz des Cosinussignals z.B. auf 0,7 Hz verringert wird, so daß keine ganze Anzahl von Perioden mehr ins Abtastfenster paßt.

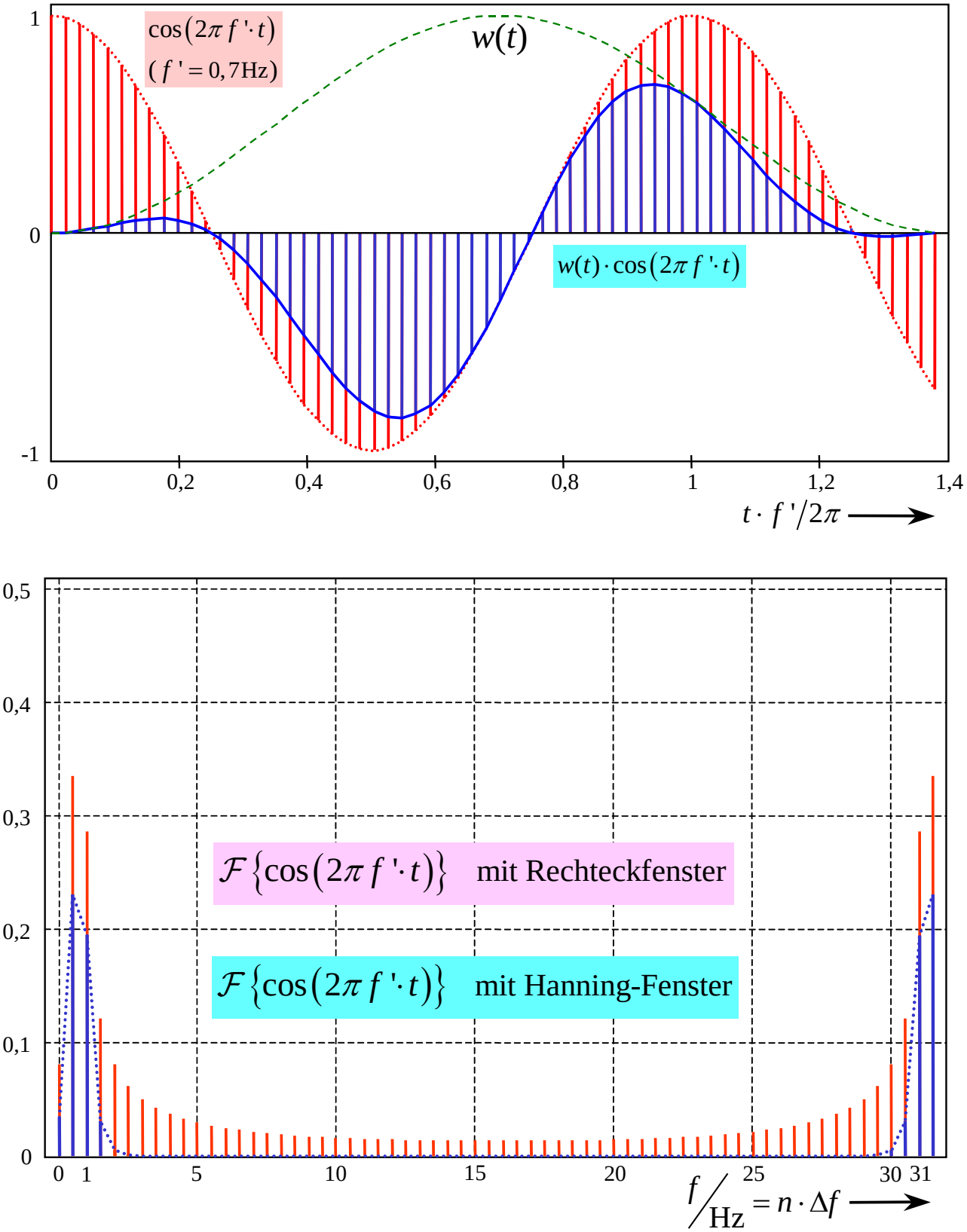


Bild 5.19: Fehlerhafte Fensterung mit Rechteck- und Hanning–Fenster im Vergleich

In der oberen Hälfte von Bild 5.19 sieht man zum einen das gedehnte Cosinussignal, von dem nun nicht mehr 2 Perioden ins Abtastfenster passen (in Rot dargestellt), und zum anderen das mit dem gestrichelt eingezeichneten Hanning-Fenster $w(t)$ gewichtete Signal (in Blau). Im unteren Bildteil sieht man, daß für das mit dem Rechteckfenster erfaßte Signal ein breites Spektrum suggeriert wird. Es ist zwar noch ein gewisser spektraler Schwerpunkt um die gewünschten Linien herum vorhanden, aber es werden zahlreiche auch weitab gelegene Linien irrtümlich erzeugt. Auch wird ein beträchtlicher Gleichanteil bei f=0 vorgetäuscht. Durch Anwendung des Hanning–Fensters gelingt es zwar nicht, die idealen Verhältnisse aus Bild 5.18 wieder herzustellen, jedoch ergibt sich eine weitgehende Unterdrückung der unerwünschten Spektrallinie, vor allem derjenigen, die relativ weit von den Sollwerten wegliegen. Um die Sollwerte bildet sich dagegen eine Art Schwerpunkt, so daß man bei einem unbekannten Signal mit diesem Ergebnis hinsichtlich des Frequenzgehaltes deutlich weniger getäuscht wird als im Fall der Rechteckfensterung.

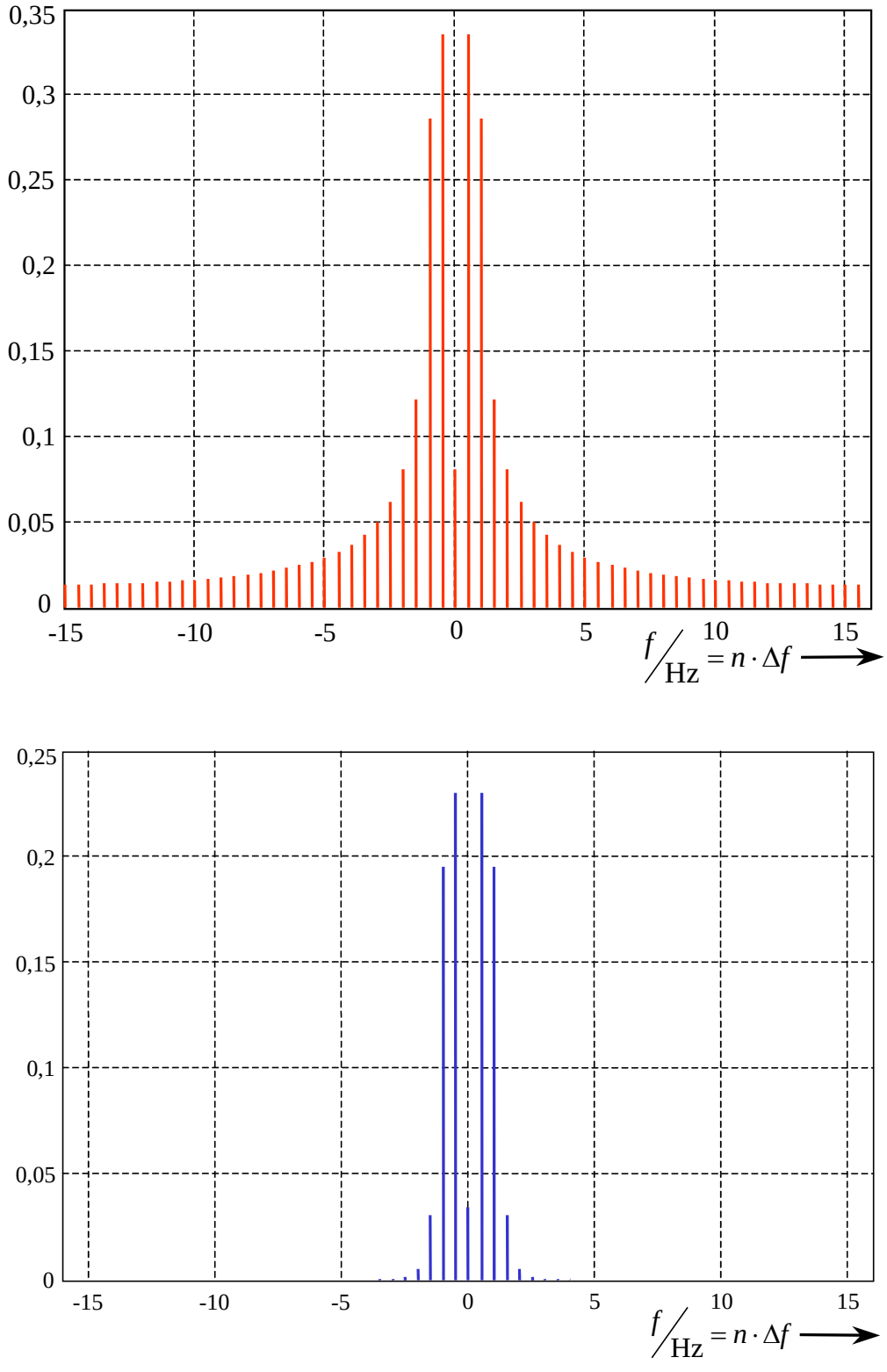


Bild 5.20 verdeutlicht nochmals den Sachverhalt.

Bild 5.20: Die beiden Spektren aus Bild 5.19 in der ’gewohnten’ Darstellung mit positiven und negativen Frequenzen (oben: Spektrum mit Rechteckfensterung; unten: Spektrum mit Hanning–Fenster)

5.8 Vergleich DFT / digitale Korrelation am Beispiel von FSK, FH und OFDM

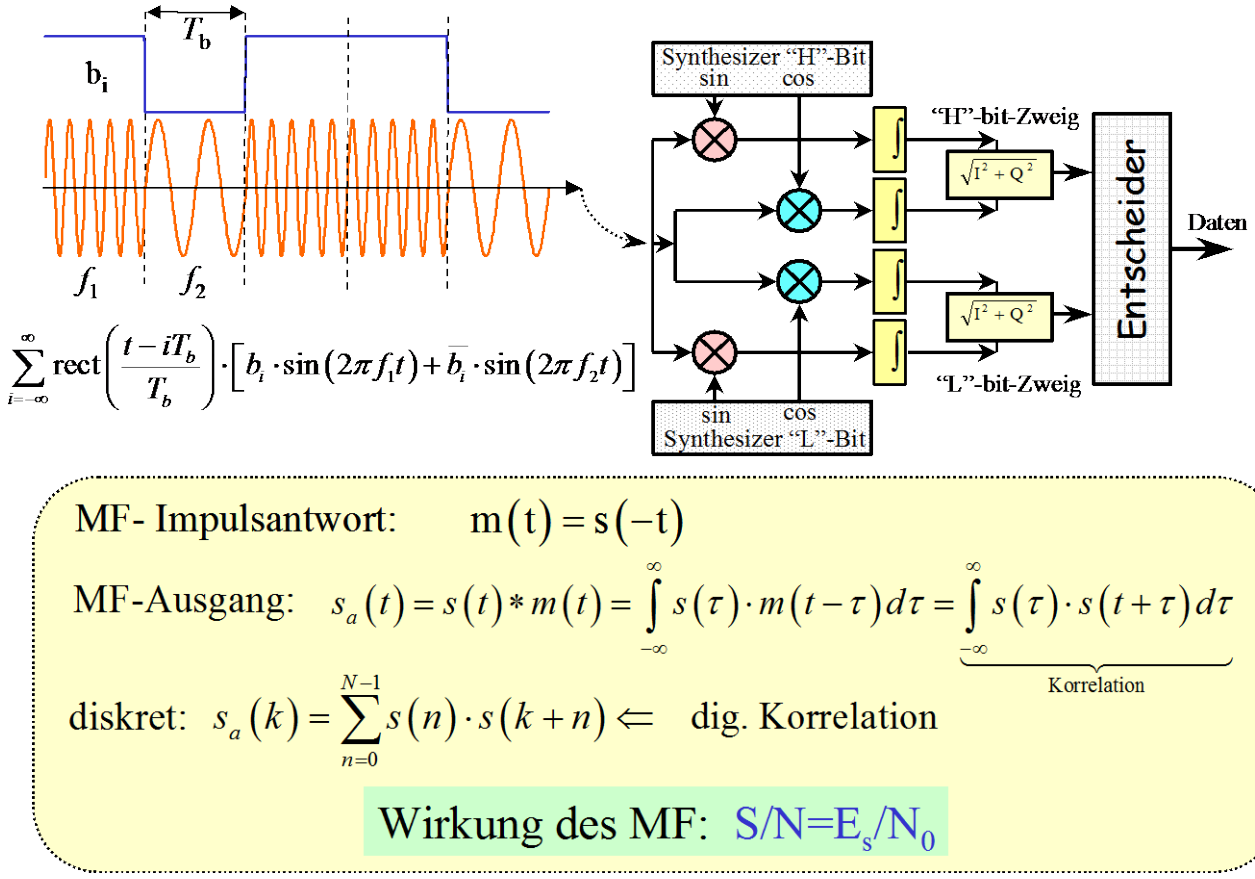


Bild 5.21: Korrelationsempfänger für ein inkohärentes FSK-Signal

Das Prinzip der digitalen Korrelation wurde bereits anhand von Bild 1.8 und (1.1) eingeführt. In Bild 5.21 sind die Grundlagen des Korrelationsempfangs am Beispiel eines inkohärenten FSK-Signals zusammengefaßt und die prinzipielle Hardwarerealisierung ist ebenfalls dort skizziert.

Ein Korrelator hat bekanntlich die Wirkung eines signalangepaßten Filters (Matched Filter, ***MF***), wenn eine Synchronisationseinrichtung dafür sorgt, daß Empfangssignal und Referenz zeitgleich sind, d.h. in Bild 5.21 bzw. (1.1) ist k=0. Man erhält dann z.B. für ein Signal $s(t)$, das im Zeitraster T abgetastet wird, die diskrete Autokorrelationsfunktion

$$\varphi_{ss}(0)=\sum_{k=0}^{N-1}\left[s(kT)\right]^2 \qquad (5.104)$$

Die Arbeitsweise eines kompletten digitalen FSK-Systems wird im weiteren anhand von Bild 5.22 erläutert, wobei der Schwerpunkt auf der Empfangssignalverarbeitung liegen wird, da die Sendeseite bereits in Kapitel 4 ausführlich diskutiert wurde. Das Empfangssignal kann mit der Frequenz f_L oder f_H auftreten und wird durch

$$s(t)=A\cdot\sin(2\pi f_{L/H}t+\alpha) \qquad (5.105)$$

beschrieben, wobei A die Amplitude und α ein beliebiger Nullphasenwinkel ist. Im betrachteten Beispiel soll eine Datenrate r_D=2400bit/s zugrunde gelegt werden und das Abtasten soll mit der Frequenz $f_A=1/T$=600 kHz und das Quantisieren mit 8 bit Auflösung erfolgen, so daß 128 Amplitudenstufen in Zweierkomplementdarstellung vorliegen (A=128). Somit ist das diskrete, ungestörte und ungedämpfte Empfangssignal gegeben durch

$$s(k) = \lceil 128 \cdot \sin(2\pi f_{L/H} kT + \alpha) \rceil . \qquad (5.106)$$

In (5.106) bedeutet $\lceil . \rceil$ ganzzahliger Anteil von (·).

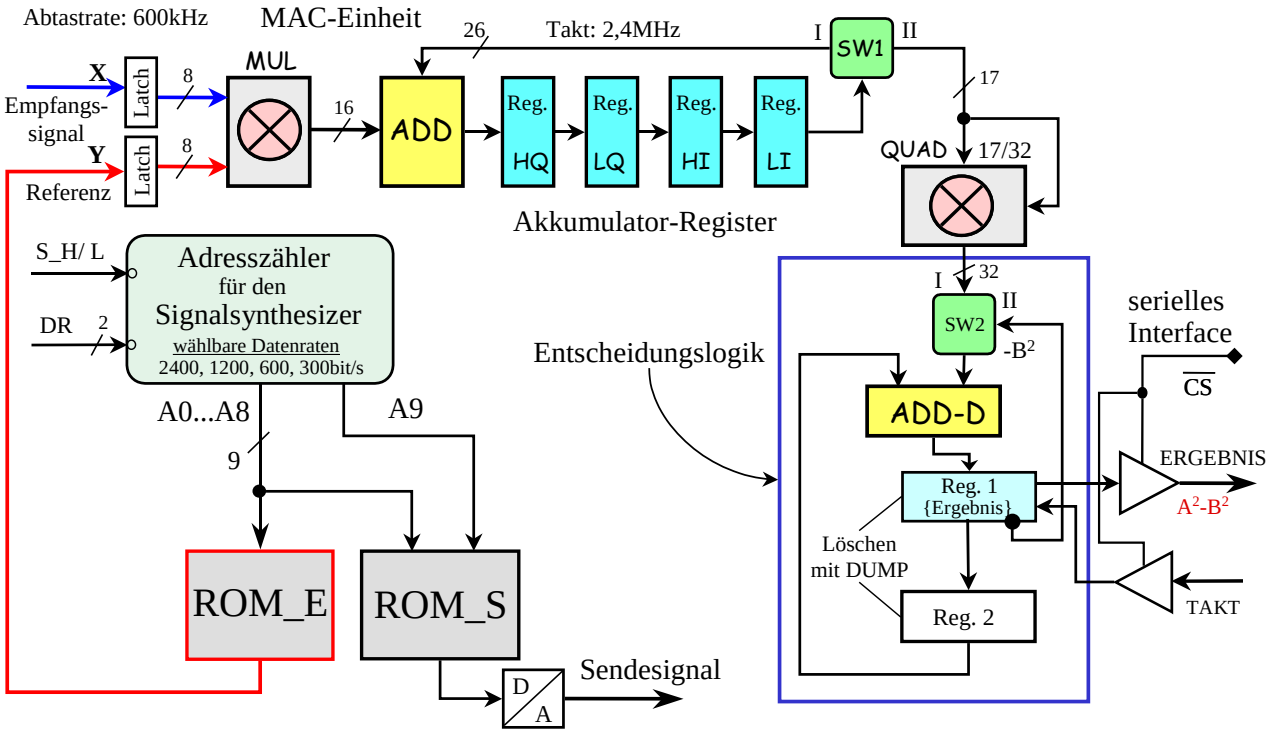


Bild 5.22: Komplettes digitales FSK-Systemkonzept für die Realisierung auf einem ASIC

Mit der Bitdauer T_B=1/2400s und der Abtastfrequenz f_A=600 kHz ergibt sich die Zahl der Abtastwerte zu $N = T_B/T$=250. Zunächst sei der Nullphasenwinkel α zu Null angenommen. Dann liefert (5.104)

$$\varphi_{ss}(0) = \sum_{k=0}^{N-1} \lceil 128 \cdot \sin(2\pi f_{L/H} kT) \rceil^2 = \frac{1}{2} \cdot 128^2 \cdot N = 2.048.000 \equiv 1F4000H \qquad (5.107)$$

Wie man sieht kann das Ergebnis mit 21 bit dargestellt werden. Da jedoch auch negative Zahlen auftreten können, wenn α nicht mehr Null ist, ist eine Zweierkomplementdarstellung sinnvoll, die dann mindestens 22 bit erfordert. Um eine gewisse Flexibilität hinsichtlich größerer Bitdauern zur Verfügung zu haben, wurde im Hardwarekonzept nach Bild 5.22 eine Registerwortbreite von 26 bit zugrunde gelegt. Damit können 2000 Abtastwerte, bzw. Bitdauern bis 3,33ms erfaßt werden.

In der Praxis kann der Nullphasenwinkel α beliebige Werte im Bereich 0...360° annehmen, die dem Empfänger nicht bekannt sind. Jetzt muß die bereits in Bild 5.21 vorgegebene Quadraturempfängerstruktur zum Einsatz kommen, wobei die Referenz jeweils in Sinus- und in Cosinuslage zur Korrelation verwendet wird und die beiden Ergebnisse geometrisch zu addieren sind. Damit ist (5.107) wie folgt zu modifizieren:

$$\frac{\varphi_{ss}(0)}{128^2} = \sqrt{\left[\underbrace{\sum_{k=0}^{N-1} \sin(2\pi f_{L/H} kT + \alpha) \cdot \sin(2\pi f_{L/H} kT)}_{\text{Inphase-Anteil}}\right]^2 + \left[\underbrace{\sum_{k=0}^{N-1} \sin(2\pi f_{L/H} kT + \alpha) \cdot \cos(2\pi f_{L/H} kT)}_{\text{Quadratur-Anteil}}\right]^2} \qquad (5.108)$$

Die grafische Auswertung von (5.108) ist im folgenden Bild 5.23 dargestellt.

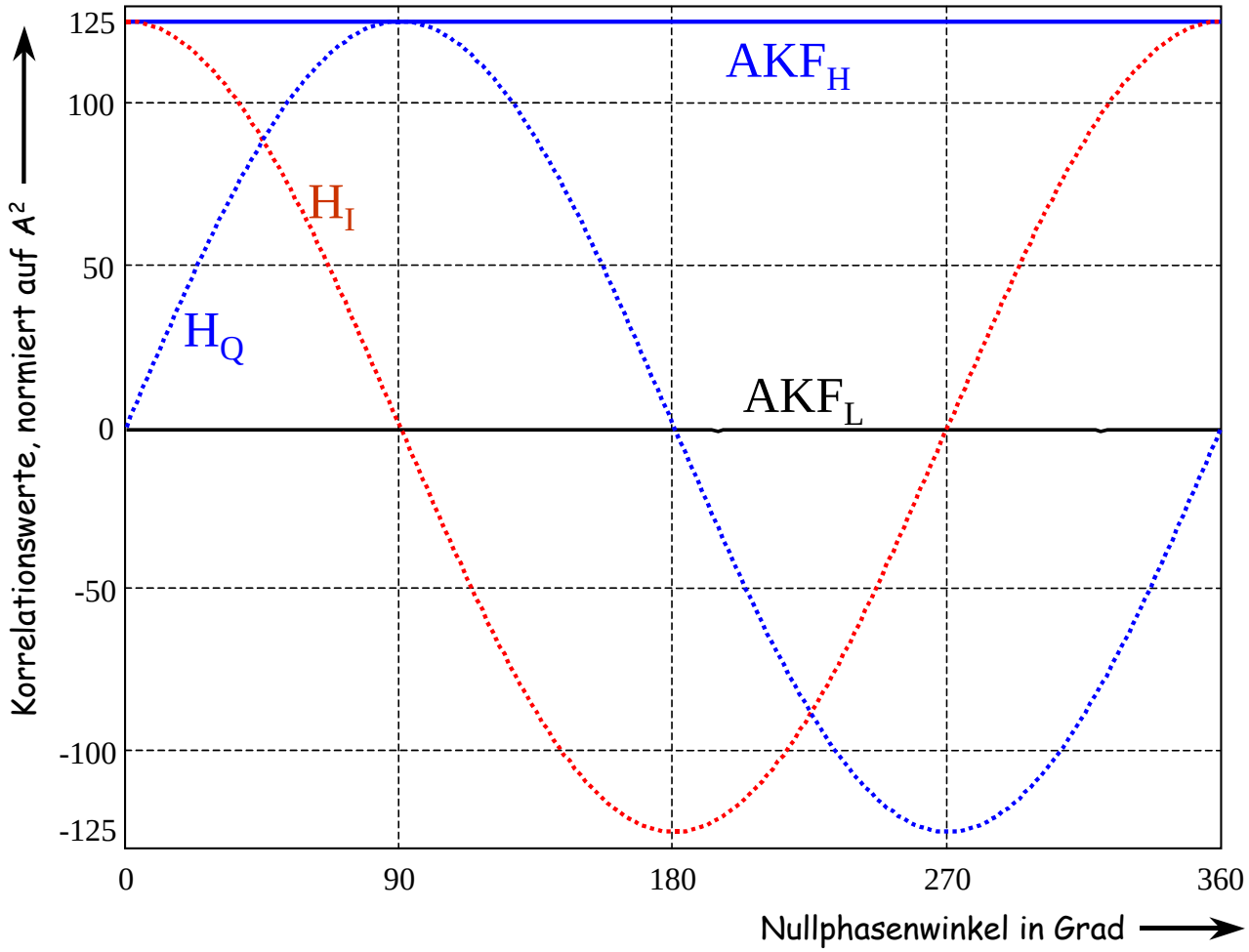


Bild 5.23: AKFs für H- und L-Signalform unter der Annahme, daß ein H-Bit gesendet wurde mit Inphase- und Quadraturanteil

Wegen der Normierung auf A^2=128^2 bewegen sich alle Werte in Bild 5.23 im Bereich $\pm N/2 = \pm 125$. Wie man sieht ist für α=0 nur der Inphaseanteil vorhanden, der mit wachsendem α gemäß einer Cosinusfunktion verläuft, d.h. bei α=90° verschwindet er, nimmt dann bei 180° das negative Maximum an und erreicht schließlich bei 360° wieder den Ausgangwert für α=0. Komplementär dazu verläuft der Quadraturanteil gemäß einer Sinusfunktion, so daß die geometrische Summe für alle α konstant den Korrelationswert $AKF_H \equiv 1/2N \cdot A^2$ nach (5.107) annimmt. Die L-Bit-Korrelation liefert aufgrund der perfekten Orthogonalität den Wert Null für alle α. Bild 5.23 verdeutlicht auch, daß eine Zweierkomplementdarstellung in den vier Registern (HQ, LQ, HI, LI) in Bild 5.22 erforderlich ist, obwohl die AKF in jedem Fall positiv ist.

Bevor der Zusammenhang mit der DFT (FFT) hergestellt wird, ist noch die Arbeitsweise der Schaltungsteile zur geometrischen Addition und Bitentscheidung in Bild 5.22 zu erläutern. Man erkennt dort vor und hinter dem Quadrierer jeweils einen Schalter (SW1, SW2). Während des Empfangs einer zu einem Bit gehörenden Signalform sind beide Schalter in Position I, so daß der Quadrierer abgetrennt ist. Sobald eine Signalform der Bitdauer T_B komplett empfangen wurde, wird SW1 für genau vier Takte in Stellung II gebracht. Dadurch werden die vier Teilkorrelationsergebnisse LI, HI LQ, HQ genau in dieser Reihenfolge quadriert und die Quadrate an die nachfolgende Entscheidungslogik weitergeleitet, deren Funktion nun beschrieben wird. Auf den Addierer ADD gelangen – da SW1 den Rückkoppelweg aufgetrennt hat - während der vier Takte Nullen, so daß in die Register die ersten vier Korrelationsprodukte HQ, LQ, HI, LI der nächsten Signalform eingeschrieben wer-

den. Dann geht SW1 wieder in Position I zurück und schließt den Rückkoppelweg für die Akkumulation. Zur Beschreibung der Entscheidungslogik wird eine schrittweise Darstellung gemäß Bild 5.24 verwendet.

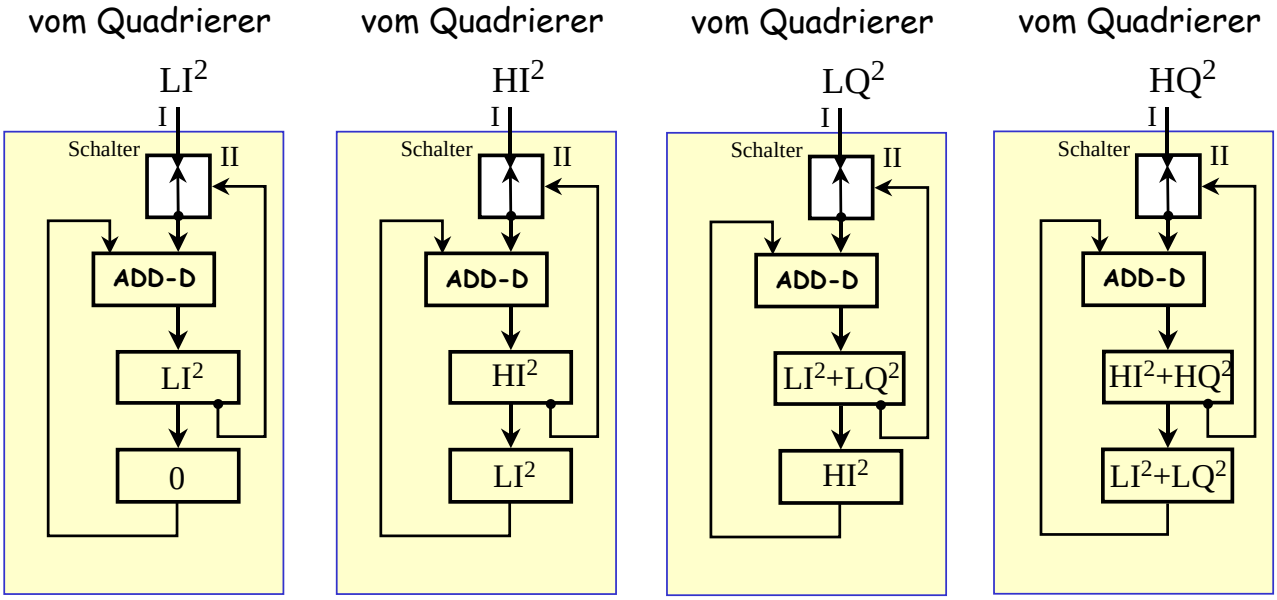


Bild 5.24: Vier Ablaufschritte zur Beschreibung der Funktion Entscheidungslogik in Bild 5.22

Wie man sieht, besteht die Entscheidungslogik aus einem Addierer und zwei Registern. Während den vier bereits oben erwähnten Taktschritten verbleibt der Schalter SW2 in Position I, so daß die quadrierten (in der Wortbreite auf 17 reduzierten) Registerinhalte nacheinander auf den Addierer ADD-D gelangen. Anhand von Bild 5.24 ist leicht nachvollziehbar, daß so am Ende des vierten Schrittes die beiden Register die quadrierten Korrelationsergebnisse H^2 und L^2 enthalten. Für die nun folgende Bitentscheidung ist es nicht nötig, die Wurzelberechnung gemäß (5.108) vorzunehmen; es kann ohne weiteres auch mit den quadrierten Ergebnissen gearbeitet werden, was natürlich eine ganz erhebliche Vereinfachung bringt – s. Bild 5.25. Der Schalter SW2 wird dazu in Position II gebracht, so daß der Addierer ADD-D neben L^2 da Einerkomplement von H^2 erhält (die Addition von Eins kann den hier zur Debatte stehend großen Zahlen vernachlässigt werden).

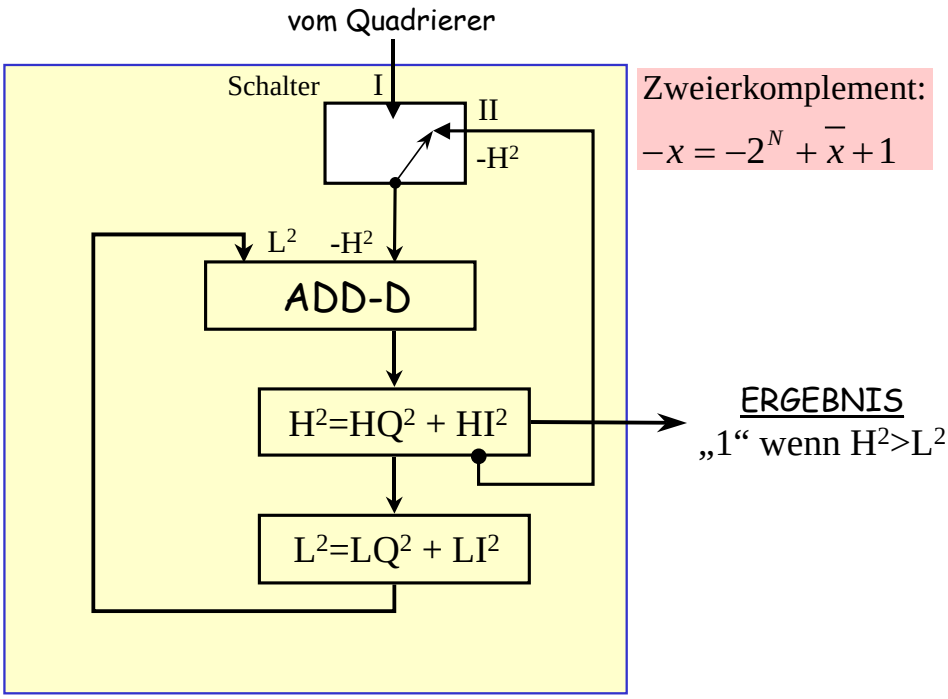


Bild 5.25: Bitentscheidung durch Berechnung der Differenz L^2-H^2

Dadurch, daß die Differenz L^2-H^2 berechnet wird und vom Addierer Ergebnisse in Zweierkomplementdarstellung geliefert werden, steht das Entscheidungsergebnis unmittelbar in Form des MSB im Ergebnisregister (Reg. 1) zur Verfügung. Falls nur eine sogenannte 'Hard-Decision' benötigt wird, muß nur dieses Bit ausgelesen werden. Für den Fall einer 'Soft-Decision', die eine feinstufige Untersuchung und auch Bewertung der 'Bitqualität' erlaubt kann das gesamte 'Entscheidungswort' in einer Länge von 33 bit gelesen werden. Soft-Decision kann z.B. bedeuten, daß eine bestimmte, betragsmäßig nicht zu kleine Differenz zwischen L^2 und H^2 vorhanden sein muß, damit die Bitentscheidung als gültig akzeptiert wird. Wenn z.B. in Sendepausen nur Rauschen empfangen wird, füllen sich alle Register in Bild 5.22 ungefähr mit gleichen Werten auf. Reine Hard-Decision würde dann stets einen zufälligen Datenstrom liefern, der den Beginn einer echten Datenübertragung maskieren würde. Die Problematik der Detektion des Beginns einer Datenübertragung auf gestörten Kanälen wird an anderer Stelle noch vertieft.

Zur Wortlänge in der Entscheidungslogik ist noch zu bemerken, daß hier gewisse Freiheitsgrade bestehen. Eine Rundung auf 17 bit ermöglicht auch noch den fehlerfreien Einsatz von Signalformen, die deutlich kürzer sind als T_B=1/2400s=416,6µs. Nach dem Quadrieren kann die Wortlänge auf 32 bit begrenzt werden, weil hier nur noch positive Zahlen vorkommen können. Die beiden Register in der Entscheidungslogik müssen dann – obwohl es bei der Addition der I- und Q-Anteile nie einen Überlauf geben kann (vgl. auch Bild 5.23) - dennoch 33 bit fassen können, da wegen der Differenzbildung wieder eine Zweierkomplementdarstellung nötig ist.

Nun wird der Zusammenhang mit der DFT (FFT) hergestellt, was letztlich bedeutet, daß die empfängerseitige Signaldetektion im Frequenzbereich statt – wie bei der Korrelation - im Zeitbereich durchgeführt wird. Diese Vorgehensweise ist bei Multicarriersystemen – insbesondere bei OFDM – die Regel. Es gibt aber auch einen technisch höchst interessanten Zwischenbereich, der zwar schon in die Kategorie Multicarrier fällt, wobei aber die Zahl der Subträger nicht allzu groß ist, d.h. < 50, für den stets im Einzelnen abzuwägen ist, ob eine „Batterie" paralleler Korrelatoren oder eine DFT vorteilhafter ist.

Wenn die Korrelationsaufgabe nach (5.108) mit den Augen der DFT betrachtet wird, dann ist zunächst rein formal die Bestimmung des Spektrums $S(n)$ aus den Zeitbereichsabtastwerten $s(k)$ gemäß (5.92) bzw. (5.95) anzusetzen. Mit den Ausgangsdaten N=250, $T = 1,\overline{6}\mu s$ beträgt die Signalformdauer NT=416,6µs und die Frequenzauflösung bei der DFT ist Δf=1/NT=2400Hz, also

$$S(n) = \sum_{k=0}^{249} s(k) \cdot \mathrm{e}^{-\mathrm{j}2\pi nk/N}, \text{ mit } \quad n = 0,1,...,249 \tag{5.109}$$

Gesucht sind hier im Gegensatz zu einer allgemeinen vollständigen DFT nicht alle N=250 Spektrallinien, sondern exakt nur die beiden, die zu den zwei FSK-Frequenz f_H und f_L gehören. Dabei ist – unter Beachtung der Forderung nach Orthogonalität – vorausgesetzt, daß f_H und f_L sich um ein ganzzahliges Vielfaches von 2400Hz unterscheiden. Zur Illustration wird im folgenden ein Beispiel mit f_L=120kHz und f_H=144 kHz=f_L+10·2400Hz gewählt.

Wegen Δf=1/NT=2400Hz ist somit für die „selektive" Berechnung der Spektrallinie bei f_L=120kHz n_L=120.000/2400=50 in (5.109) einzusetzen; entsprechend bei f_H=144 kHz n_H=144.000/2400=60.

Die Ergebnisse sind in dargestellt, wobei die bei negativen Frequenzen angesiedelten Linien – wie bei der DFT üblich – an den Stellen N–$n_{H/L}$ zu finden sind.

Die Auswertung von (5.109) führt somit auf

$$\begin{aligned} S(50) &= \sum_{k=0}^{N-1} 128\cdot\sin\left(2\pi f_L kT\right)\cdot\cos\left(2\pi\frac{50}{250}\cdot k\right) \quad \equiv \mathrm{Re}\,1 = 0 \\ &\quad -\mathrm{j}\sum_{k=0}^{N-1} 128\cdot\sin\left(2\pi f_L kT\right)\cdot\sin\left(2\pi\frac{50}{250}\cdot k\right) \quad \equiv \mathrm{Im}\,1 = \frac{1}{2}N\cdot 128 \\ S(200) &= \sum_{k=0}^{N-1} 128\cdot\sin\left(2\pi f_L kT\right)\cdot\cos\left(2\pi\frac{200}{250}\cdot k\right) \quad \equiv \mathrm{Re}\,2 = 0 \\ &\quad +\mathrm{j}\sum_{k=0}^{N-1} 128\cdot\sin\left(2\pi f_L kT\right)\cdot\sin\left(2\pi\frac{200}{250}\cdot k\right) \quad \equiv \mathrm{Im}\,2 = \frac{1}{2}N\cdot 128 \end{aligned} \tag{5.110}$$

Hierbei wegen ist $f_L = n_L\cdot\Delta f$ das Produkt $f_L\cdot T = T\cdot n_L/N\cdot T = n_L/N$ (=50/250), so daß aufgrund der Orthogonalität die Realteile in (5.110) verschwinden und die Imaginärteile maximal werden.

Man beachte, daß im Gegensatz zur Korrelation nach (5.107) nicht A^2 sondern nur A im Produkt der Sinusschwingungen auftritt. Das ergibt sich aus der formalen Anwendung von (5.109), wo die e-Funktion mit Eins und nicht mit A gewichtet ist. Durch Normierung der Ergebnisse auf A statt A^2 erhält man die in Bild 5.26 dargestellten Werte, die mit denen aus Bild 5.23 vergleichbar sind.

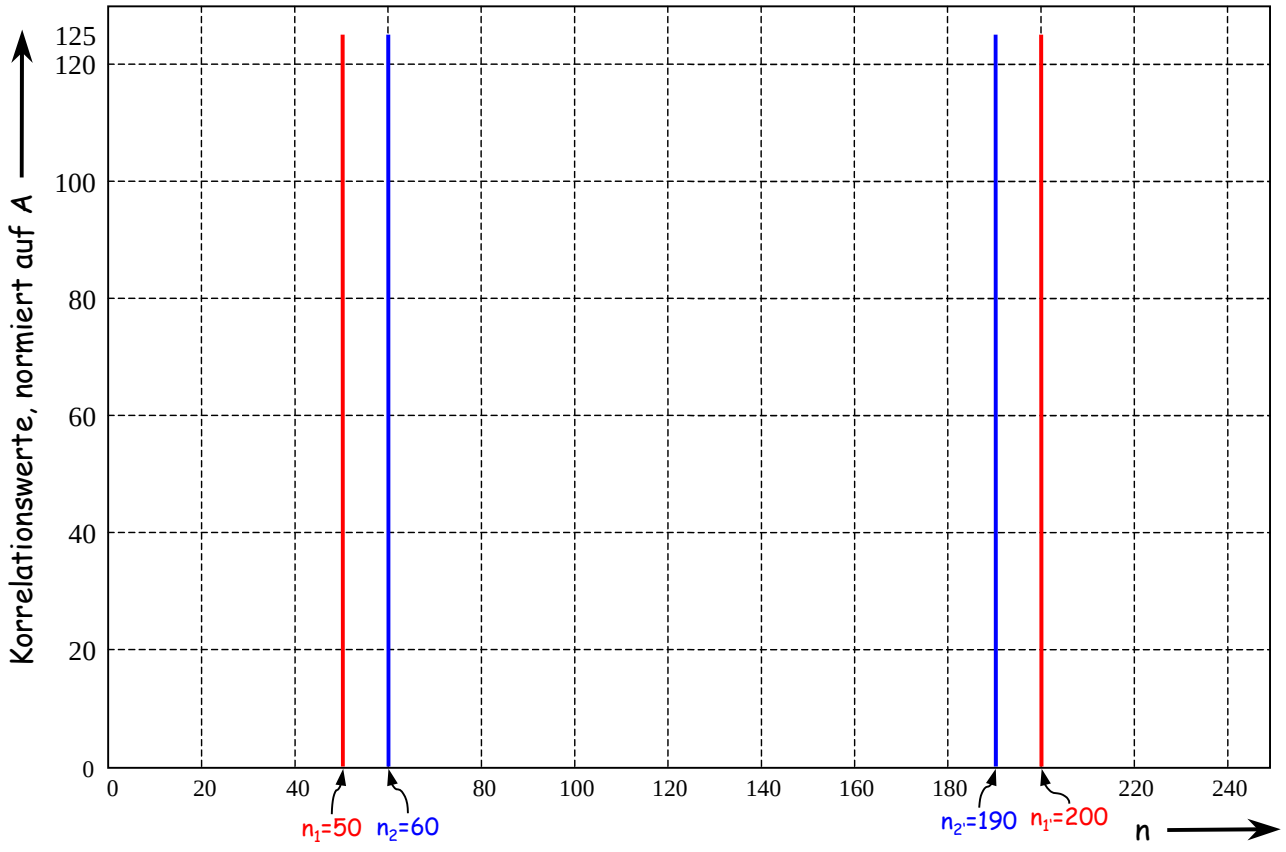


Bild 5.26: 'Selektive' DFT zur FSK–Signaldetektion

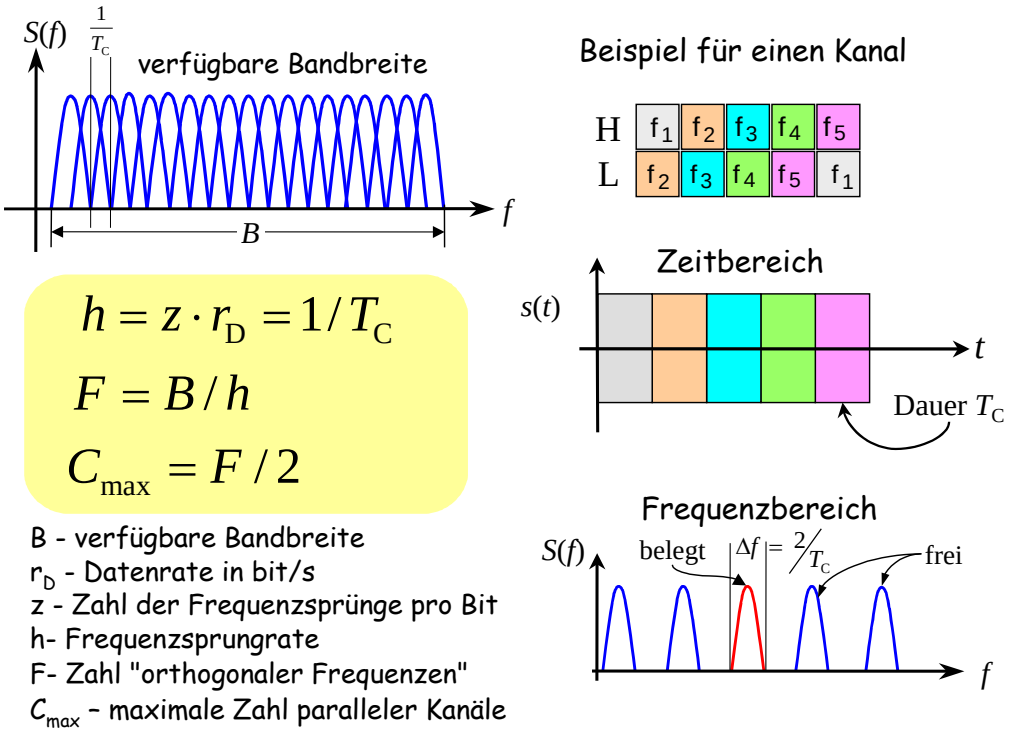
Der Zusammenhang zwischen Zeitbereich und Frequenzbereich läßt sich nun unmittelbar über das Parsevalsche Theorem – siehe (5.36) – herstellen. Danach ist zur Ermittlung der Signalformenergie, die bekanntlich dem Maximum der AKF $\varphi_{ss}(0)$ entspricht, eine Summation über das Leistungsdichtespektrum durchzuführen, d.h. es gilt

$$\varphi_{ss}(0) = \sum_{k=0}^{N-1}\left[s(k)\right]^2 = \frac{1}{N}\sum_{n=0}^{N-1}\left|S(n)\right|^2 \quad \equiv \text{ Maximum der AKF .} \tag{5.111}$$

Mit (5.110) erhält man dann wegen Re1=Re2=0 und Im1=Im2=1/2·N·128^2:

$$\begin{aligned}\varphi_{ss}(0) &= \frac{1}{N}\sum_{n=0}^{N-1}\left|S(n)\right|^2 = \frac{1}{N}\left[\mathrm{Re}1^2 + \mathrm{Im}1^2 + \mathrm{Re}2^2 + \mathrm{Im}2^2\right] \\ &= 2\cdot\frac{1}{N}\left\{\mathrm{Im}1\right\}^2 = \frac{2}{N}\left[\frac{1}{4}N^2\cdot 128^2\right] = \frac{N}{2}\cdot 128^2\end{aligned} \tag{5.112}$$

Wie man sieht, entspricht das Ergebnis exakt dem der diskreten Korrelation nach (5.107).



$$h = z \cdot r_D = 1/T_C$$
$$F = B/h$$
$$C_{max} = F/2$$

B - verfügbare Bandbreite
r_D - Datenrate in bit/s
z - Zahl der Frequenzsprünge pro Bit
h- Frequenzsprungrate
F- Zahl "orthogonaler Frequenzen"
C_{max} - maximale Zahl paralleler Kanäle

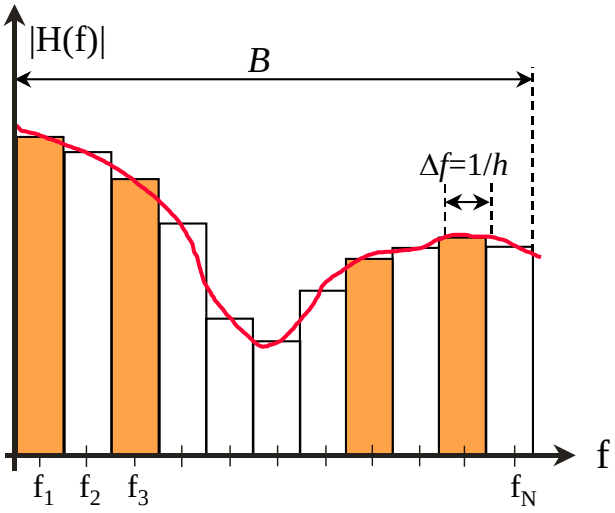
Die Verallgemeinerung hin zum Einsatz sehr vieler Träger ist in Bild 5.29 skizziert.

Obwohl das skizzierte Spektrum $S(f)$ schon sehr nach OFDM aussieht, gibt es zwei wesentliche Unterschiede, nämlich, daß bei FH die Träger nicht parallel sondern zeitlich nacheinander gesendet werden und daß sie nicht moduliert sind.

Bild 5.27: Erweiterung von FH zu hohen Trägerzahlen

Vor dem Übergang zu OFDM soll noch kurz ein modifiziertes FH–System betrachtet werden, das für die störresistente Kommunikation über Stromnetze – insbesondere zur Zählerfernabfrage und zur Gebäudeautomatisierung – mit Datenraten bis 2400 bit/s konzipiert wurde.

Nutzung von N Subkanälen =>N! mögliche Symbole
- werden nur N Symbole verwendet, sind diese in allen Frequenzen verschieden => maximale Störresistenz
- jedes Symbol überträgt ld(N) bit



optimale Symbolverarbeitung für N=4

Frequenzfolge	Datensymbole
f_1 f_2 f_3 f_4	LL
f_4 f_1 f_2 f_3	LH
f_3 f_4 f_1 f_2	HL
f_2 f_3 f_4 f_1	HH

f_1	f_2	f_3	f_4
52800Hz	62400Hz	72000Hz	86400Hz

Empfangssignalverarbeitung:
- inkohärenter Korrelationsempfang;
- Akkumulation der Symbolenergie (soft decision)
- Verwenden von Zuverlässigkeitsinformation

Bild 5.28: Konzept eines modifiziertes FH–Systems mit vier Trägern

Die Besonderheit des Konzeptes nach Bild 5.30 ist ein Abweichen vom Prinzip der einfachen Mehrheitsentscheidung im Empfänger. Es werden vier Trägerfrequenzen, die „optimal“ im verfügbaren Spektrum plaziert sind, verwendet. Insgesamt ließen sich so 4!=24 verschiedene „Symbole“ darstellen. In gestörter Umgebung wäre eine solche Datenübertragung aber nicht besonders zuverlässig, weil der Ausfall einer Frequenz – sei es durch hohe selektive Dämpfung oder schmalbandige Störung – sofort Bitfehler nach sich ziehen würde. Wenn dagegen nur ld(N)=2 bit in ein Symbol gepackt werden, dann erhält man – natürlich auf Kosten der Datenrate – ein Optimum an Störresistenz, d.h. es können bis zu drei der vier Trägerfrequenzen ausfallen, ohne daß ein Bitfehler auftritt.

Das Konzept nach Bild 5.30 wurde im Zeitraum 1995-1997 in Zusammenarbeit mit der Industrie in einem ASIC realisiert, dessen Aufbau und praktischer Einsatz in den folgenden vier Bildern illustriert ist.

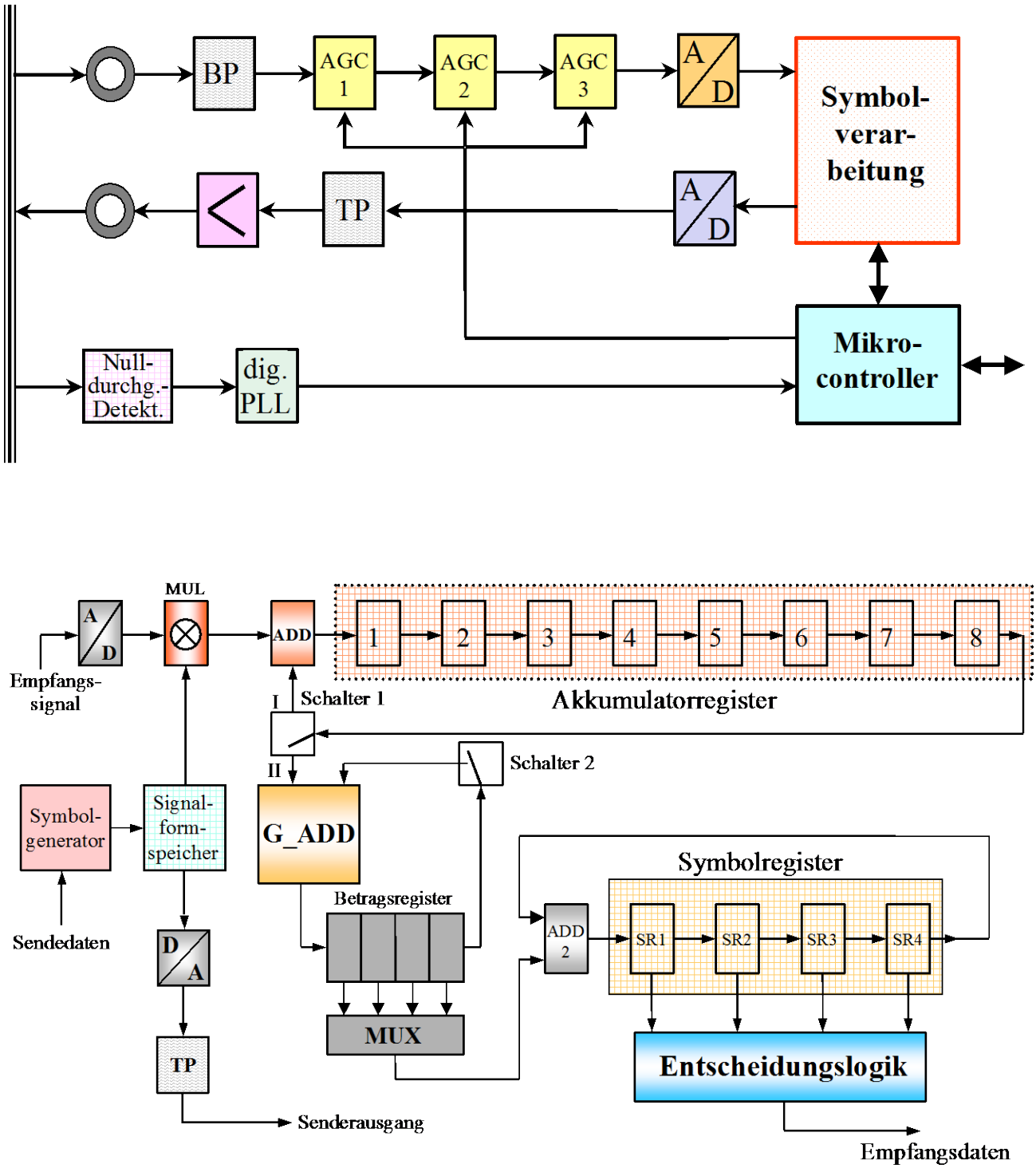


Bild 5.29: Modifiziertes FH–System für vier Träger. Oben: Blockschaltbild; unten: Funktioneller Aufbau zur Implementierung in einem ASIC

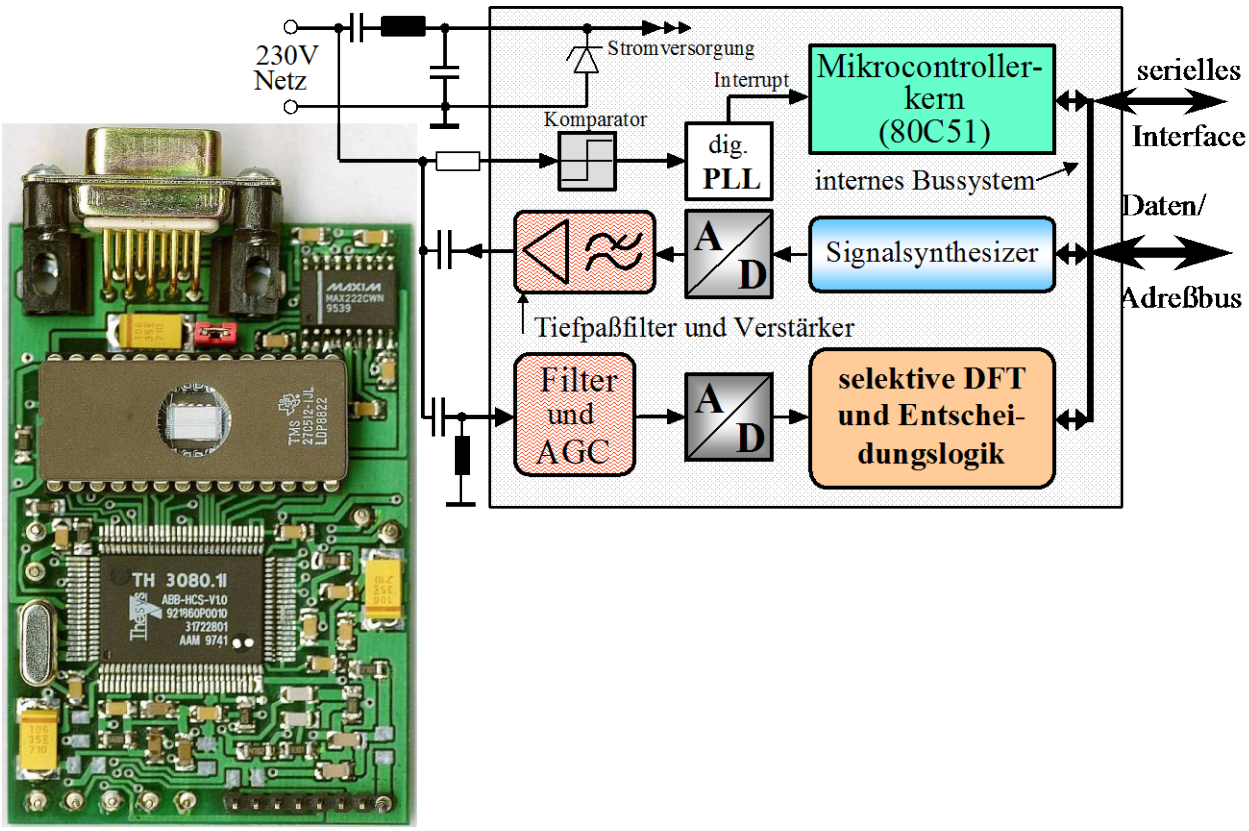


Bild 5.30: Realisiertes ASIC im Prototyp (oben) und im industriellen Seriengerät (unten)

5.8.1 Kommunikation mit hohen Datenraten über ungewöhnliche Kanäle - OFDM

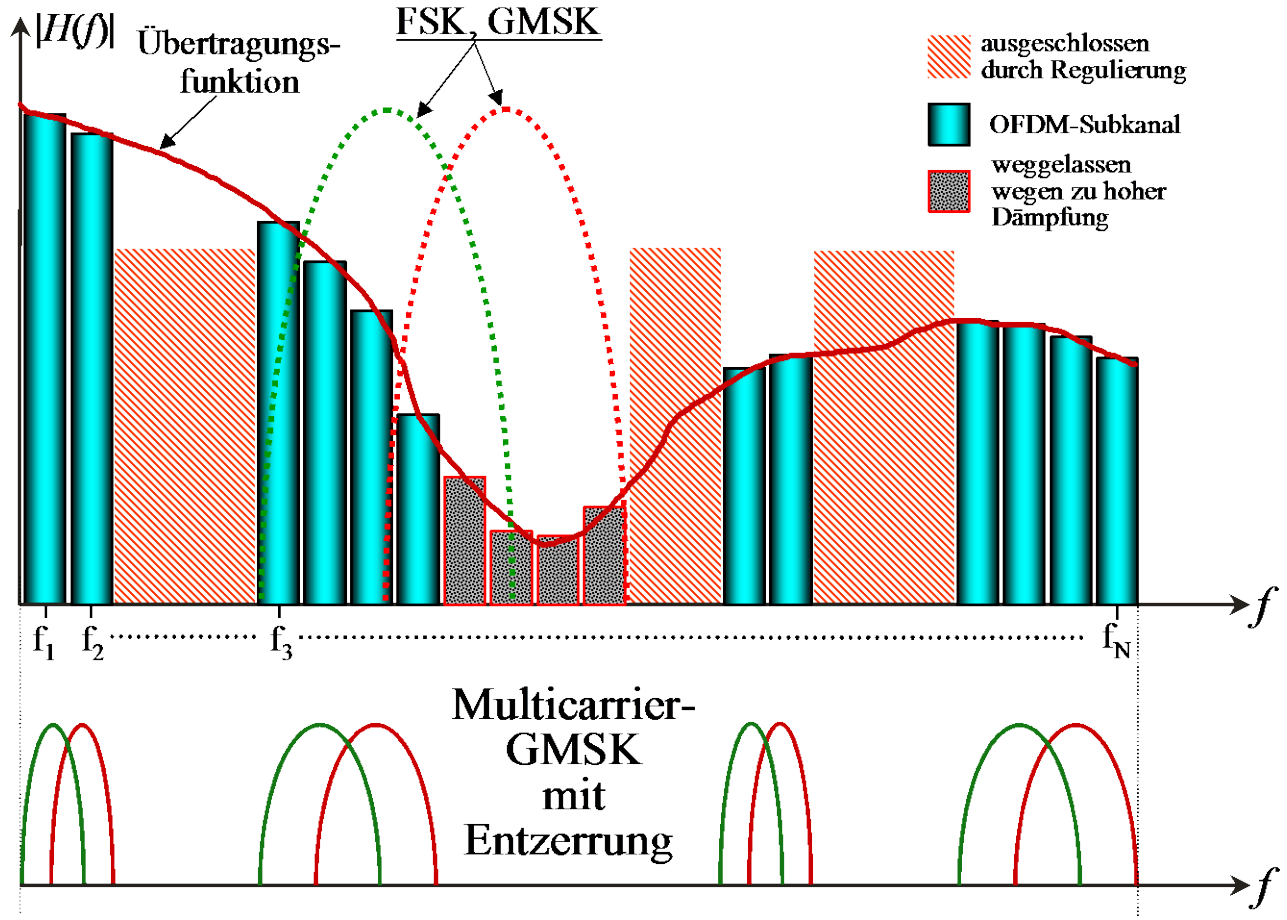


Bild 5.31: Vergleich von Möglichkeiten der schnellen Datenübertragung auf echobehafteten Kanälen

Bei echobehafteten Kanälen liegt Mehrwegeausbreitung vor, so daß das gewünschte Empfangssignal sein Ziel – den Empfänger – über verschieden lange und damit unterschiedlich stark dämpfende Pfade erreicht. Das Resultat ist die so genannte „Intersymbolinterferenz“, die so stark in Erscheinung treten kann, daß auch ohne fremde Störungen ein fehlerfreier Empfang unmöglich wird, obwohl genügend Empfangsleistung zur Verfügung steht.

Anhand von Bild 5.32 wird die Problematik verdeutlicht.

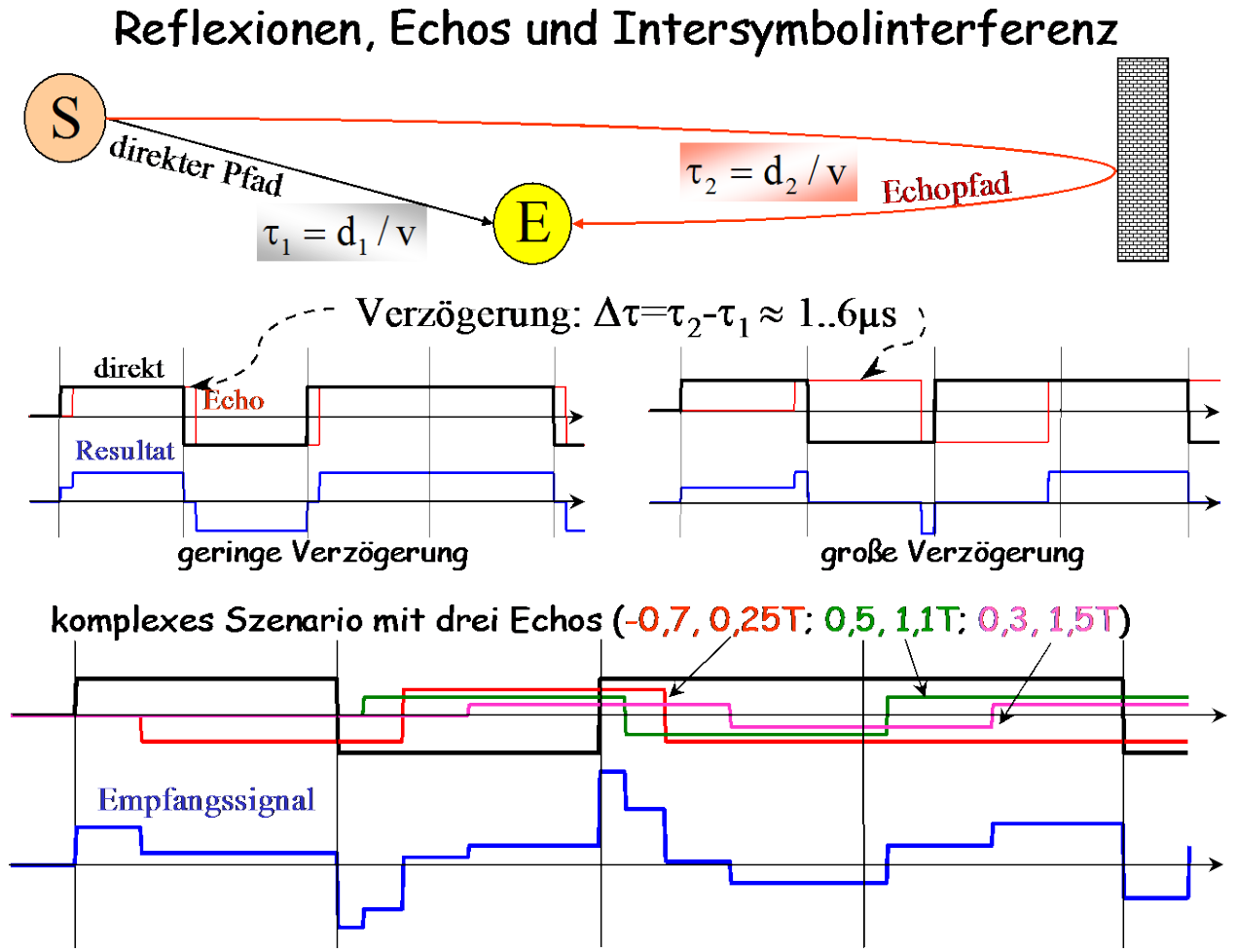
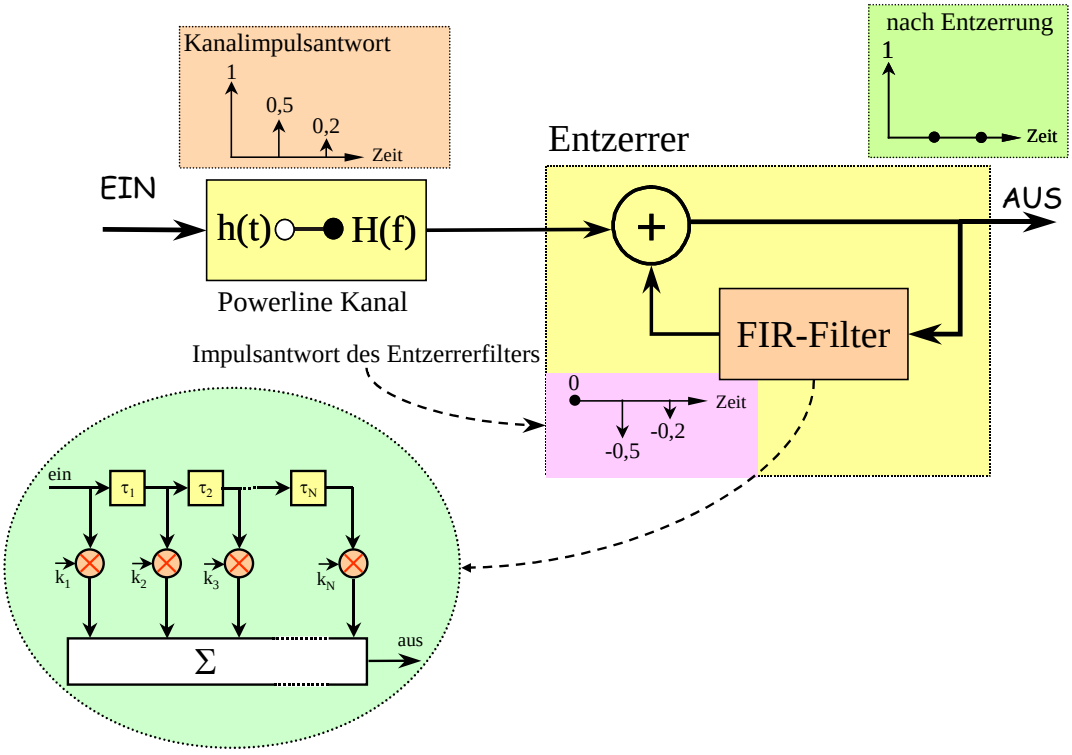


Bild 5.32: Reflexionen, Echos und Intersymbolinterferenz

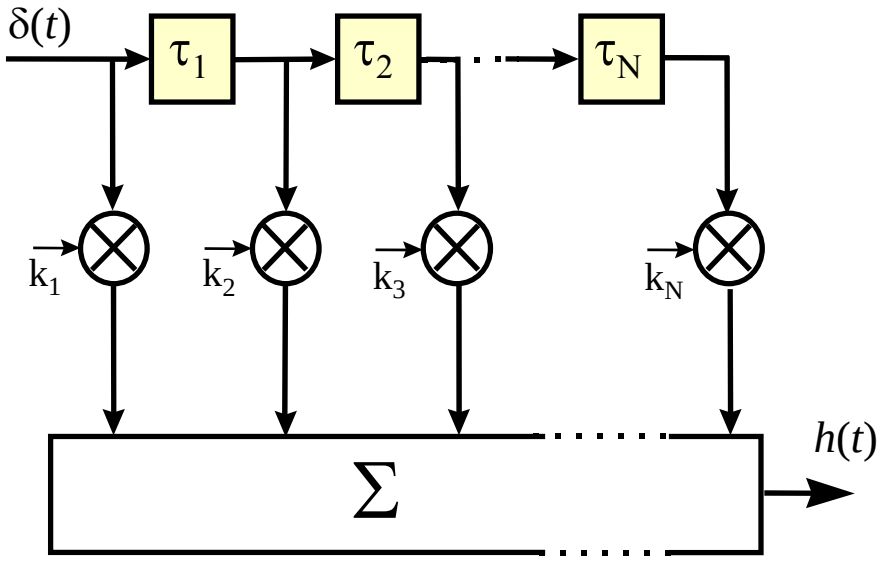


Prinzipiell können an einem echobehafteten Kanal Entzerrer eingesetzt werden, die Echos unwirksam machen, d.h. die Hintereinanderschaltung von Kanal und Echoentzerrer ergibt ein LTI-System mit idealer Impulsantwort in Form eines Diracimpulses.

Bild 5.33: Prinzip der Echoentzerrung mittels Kanalschätzung und einstellbarem FIR-Filter

Wie man in Bild 5.33 erkennt, wird die Entzerrung mittels eines einstellbaren FIR-Filters durchgeführt, das zu jedem Kanalecho gezielt ein passendes 'Gegenecho' erzeugt, so daß ein Auslöschung der Wirkung des Kanalechos erfolgt. Der Aufbau des FIR-Filters kann bei hoher Echoanzahl und je nach erforderlicher Taktrate ziemlich aufwendig werden und stellt dennoch nur einen Teil des Entzerrers dar, denn bevor die Einstellung des FIR-Filters erfolgen kann, muß eine Kanalvermessung durchgeführt werden, die Lage, Vorzeichen und Stärke der Echos ermittelt. Das kann nicht während einer Datenübertragung geschehen, weil dazu festgelegte Signalformen, die keinen Zufallscharakter haben dürfen – so genannte Pilottöne oder 'Midambeln' – gesendet werde müssen. Zu den Zeiten der Kanalvermessung ist die Datenübertragung unterbrochen, so daß eine verringerte 'effektive' Datenrate zur Verfügung steht.

Beispiel eines leitungsgebundenen Echokanals: Powerline-Kanalmodell



$$h(t)=\sum_{i=1}^{N} k_i \cdot \delta(t-\tau_i) \quad \circ\!\!-\!\!\bullet \quad H(f)=\sum_{i=1}^{N} k_i \cdot \mathrm{e}^{-\mathrm{j}2\pi f \tau_i}$$

Ein Echo mit der Verzögerung $\tau_2=T$ und dem Faktor $k_2=0{,}8$ liefert mit $k_1=1$ und $\tau_1=0$ für den direkten Pfad

$$H(f)=1+0{,}8\cdot \mathrm{e}^{-\mathrm{j}2\pi f T}$$

Betragsverlauf:

$$\begin{aligned}|H(f)| &= \sqrt{\mathrm{Re}\{H[f]\}^2+\mathrm{Im}\{H[f]\}^2} = \{1+0{,}8\cos(2\pi fT)\}^2+\{-j\cdot 0{,}8\cdot\sin(2\pi fT)\}^2 = \\ &= \sqrt{1+1{,}6\cdot\cos(2\pi fT)+0{,}64\cdot[\cos(2\pi fT)]^2+0{,}64\cdot[\sin(2\pi fT)]^2} = \\ &= \sqrt{1{,}64+1{,}6\cdot\cos(2\pi fT)}\end{aligned}$$

Logarithmisch:

$$\left|H(f)\right\|_{\log} = 20 \cdot \lg\left[\sqrt{1,64 + 1,6 \cdot \cos(2\pi fT)}\right]$$

Die Koeffizienten k_i hängen nicht nur von der Kabellänge, sondern auch von der Frequenz ab. Basierend auf zahlreichen Kanaluntersuchungen wurde für die Koeffizienten k_i der folgende Ausdruck gefunden:

$$k(f, \ell_i) = a_i \cdot \mathrm{e}^{-\alpha(f)\cdot\ell_i}$$

In steht ℓ_i für die Kabellänge und a_i ist ein spezieller Gewichtsfaktor, der Details der Netztopologie erfaßt. a_i stellt das Produkt der Reflexions- und Transmissionsfaktoren im Zuge des Pfades *i* dar. Wenn man die Effekte der Mehrwegeausbreitung und der frequenz- und längenabhängigen Dämpfung zusammenfaßt und die Phasengeschwindigkeit v_p einführt, erhält man schließlich die vollständige Übertragungsfunktion

$$H_E(f) = \sum_{i=1}^{N} a_i \cdot \mathrm{e}^{-\alpha(f)\cdot\ell_i} e^{-\mathrm{j}2\pi f\frac{\ell_i}{v_p}}.$$

Wenn man die Dämpfungskonstante $\alpha(f)$ für verschiedene Leitungsarten im Detail untersucht, kommt man zu dem Ergebnis

$$\alpha(f) = \frac{R'}{2Z_L} + \frac{G' \cdot Z_L}{2} \quad \text{und} \quad \beta = \omega\sqrt{L'C'} = \frac{\omega}{v_p}$$

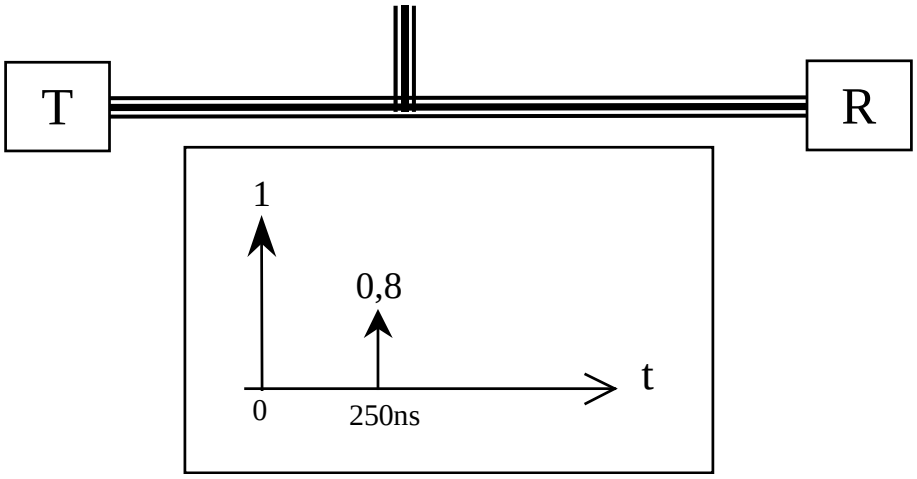
aus dem sich die Näherung

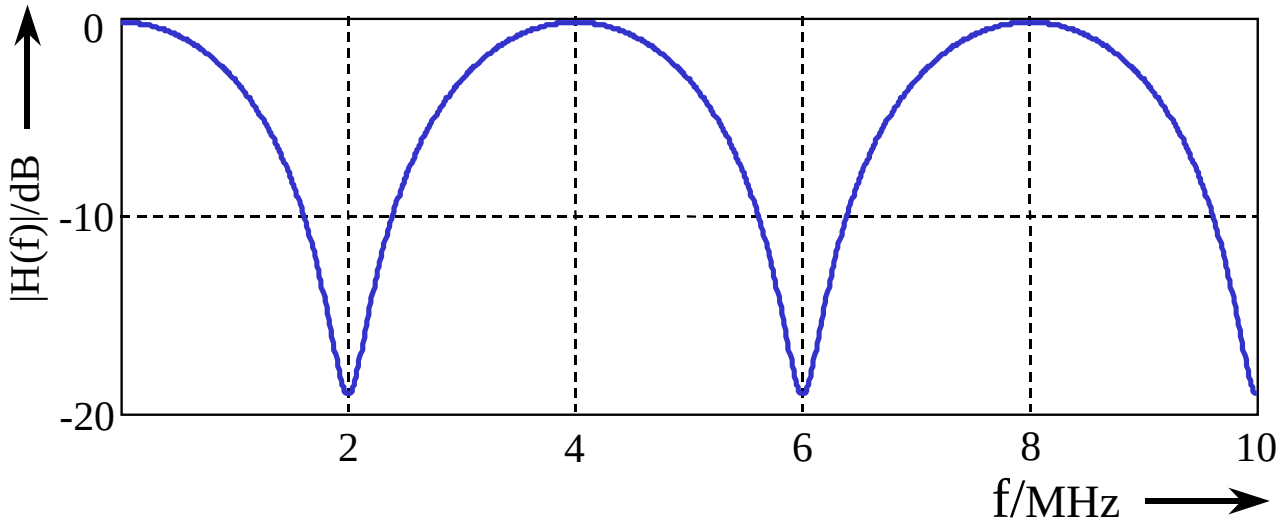
$$\alpha(f) \approx \vartheta_1 \cdot \sqrt{f} + \vartheta_2 \cdot f$$

ableiten läßt, aus der sich schließlich

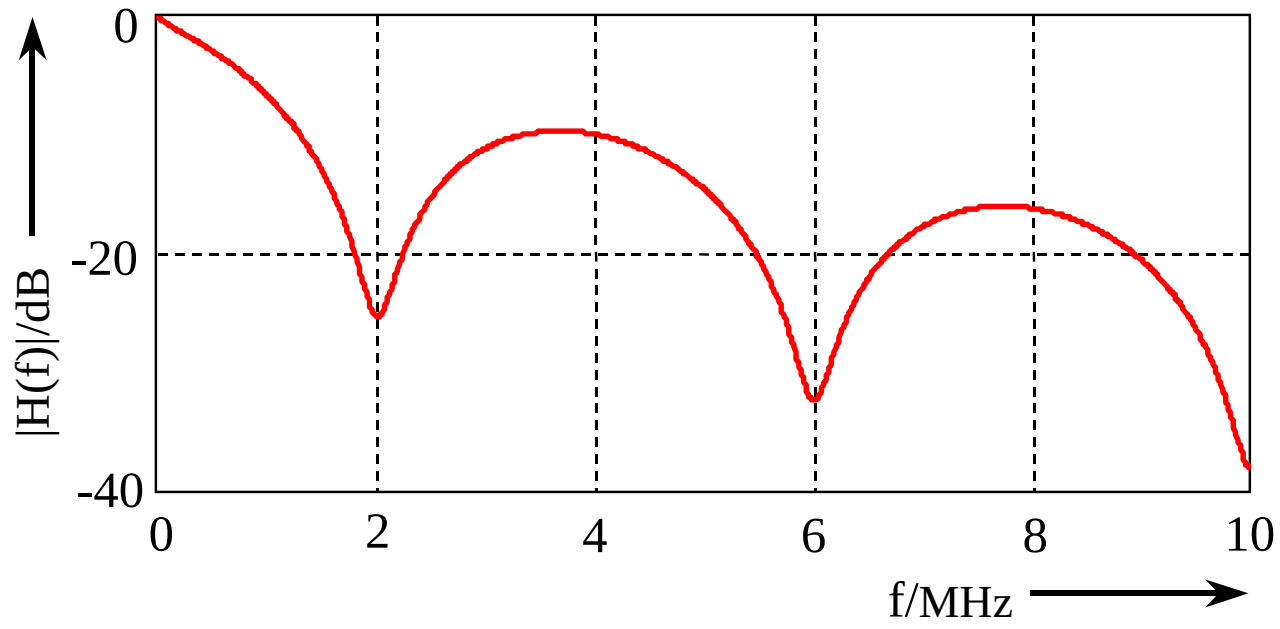
$$\alpha(f) \approx \left(\eta_0 + \eta_1 \cdot f^{\varepsilon}\right)$$

ergibt. Die Koeffizienten η_1 und η_2 und ε sind im allgemeinen für eine Kabelart konstant und ε bewegt sich in einem Bereich von 0,5...0,7.





$|H(f)|$ für T=250 ns



$H(f)|$ für T=250 ns und $\alpha(f)\cdot\ell = 0{,}017\cdot\left(\frac{f}{10\text{kHz}}\right)^{0{,}7}$, normiert auf $|H(0)|$

OFDM erlaubt eine ISI-freie Übertragung über echobehaftete Kanäle, ohne daß Details der Impulsantwort bekannt sind, d.h. eine Vermessung entfällt komplett. Der Aufwand steckt – wie im weiteren deutlich wird – in der Durchführung einer IFFT auf der Senderseite und einer FFT auf der Empfängerseite.

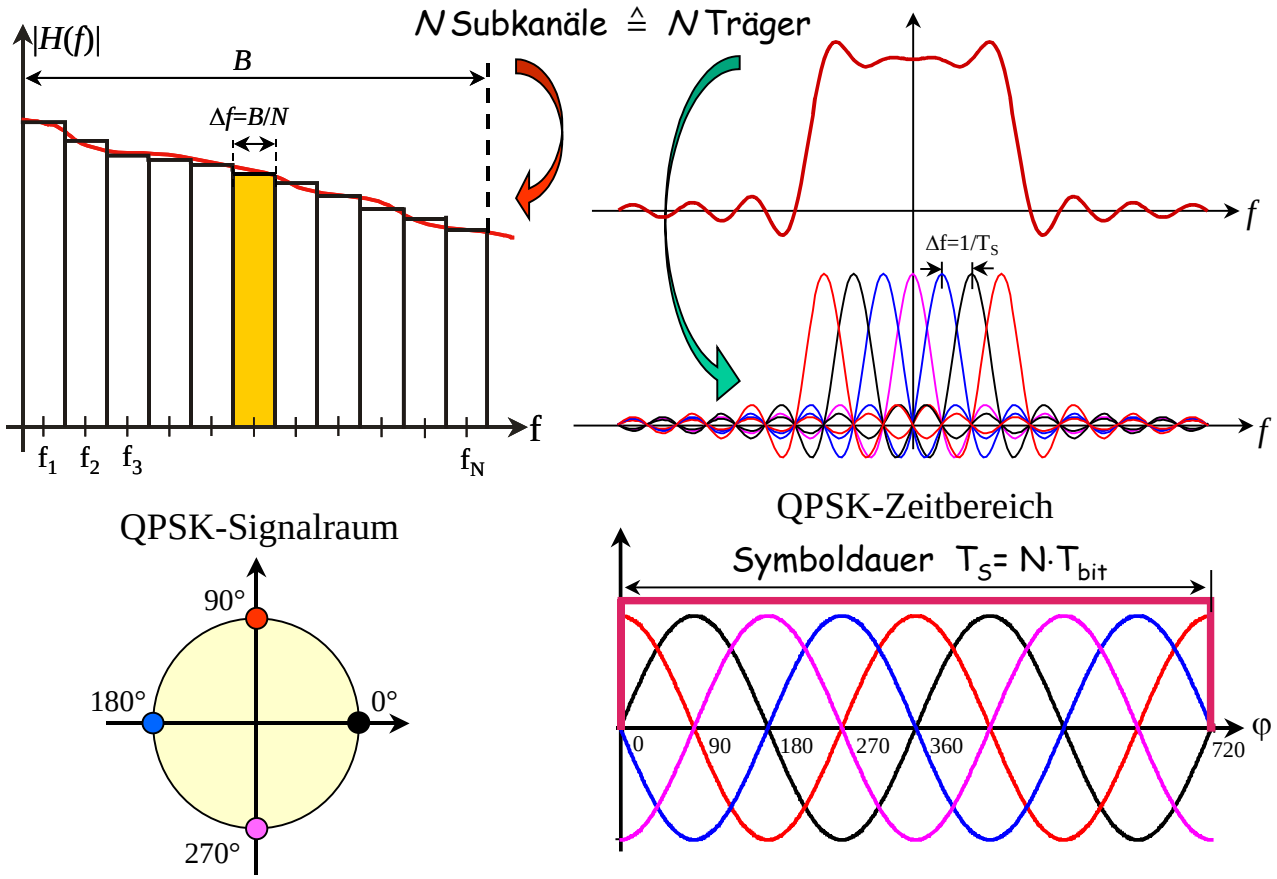


Bild 5.34: OFDM – ein Übertragungsverfahren mit „eingebauter“ ISI-Unterdrückung

OFDM ist keineswegs ein neues Modulationsverfahren, sondern wird seit geraumer Zeit schon technisch genutzt, so z.B. beim digitalen Hörrundfunk (Digital Audio Broadcasting, DAB), beim digitalen terrestrischen Fernsehen (Digital Video Broadcasting, DVBT) und bei der schnellen Datenübertragung auf Telefonleitungen (Asymmetric Digital Subscriber Line, ADSL). OFDM läßt sich in einfacher Weise aus dem behandelten bandspreizenden Verfahren „Frequency Hopping“ (FH) entwickeln. Damit hat auch OFDM ’Spread-Spectrum’-Eigenschaften, d.h. es lassen sich sowohl hohe Störresistenz, geringe Leistungsdichte und Vielfachzugriff realisieren. OFDM ist aber im Gegensatz zu Spread-Spectrum-Techniken ein Mehrträgerverfahren, bei dem das verfügbare Spektrum in zahlreiche schmale Subkanäle eingeteilt wird, wobei der Datenstrom im Frequenzmultiplex (Frequency Division Multiplexing, FDM) auf einer großen Anzahl N von Trägern mit den Frequenzen $f_1,f_2,...f_N$ parallel übertragen wird. Jeder dieser schmalbandigen Teilkanäle hat dabei die Breite

$$\Delta f = \frac{B}{N}. \tag{5.113}$$

In Bild 5.34 oben links ist dieser Sachverhalt veranschaulicht. Aufgrund des Schmalbandcharakters der Teilkanäle kann davon ausgegangen werden, daß Dämpfung und Gruppenlaufzeit innerhalb eines jeden Teilkanals nahezu konstant sind. Durch diese idealen Eigenschaften wird – sofern überhaupt nötig – die Kanalentzerrung für die Teilkanäle im Empfänger sehr einfach. Dies ist ein wesentlicher Vorteil gegenüber breitbandigen Einträgerverfahren.

Bekanntlich ist der minimal zulässige Frequenzabstand Δf bei FH gleich der Hoprate $h=1/T_c$. Man hat dann einen orthogonalen Signalformsatz der Größe N Signalen, der z.B. symmetrisch um eine in der Mitte des Übertragungsbandes der Bandbreite B liegende Mittenfrequenz f_0 gruppiert sein kann und wie folgt beschrieben werden kann:

$$s_i(t) = A \cdot rect\left(\frac{t}{T}\right) \cdot \sin\left(2\pi\left(f_0 + \left[i - \frac{N+1}{2}\right] \cdot \Delta f\right)t\right), \quad \text{mit} \quad i \in \{1 \ldots N\} \qquad (5.114)$$

$s_i(t)$ hat am unteren Bandende (i=0) die Frequenz $f_0-[(N-1)/2]\cdot\Delta f = f_0-(B-\Delta f)/2$ und am oberen Bandende (i=N) die Frequenz $f_0+[(N-1)/2]\cdot\Delta f = f_0+(B-\Delta f)/2$. In Bild 5.34 oben rechts ist dieser Sachverhalt illustriert. Man erkennt hier ganz deutlich der Bezeichnung ***orthogonal***. Die Spektren im Abstand $\Delta f=1/T_s$ überlappen sich jeweils zur Hälfte, aber alle Nullstellen fallen zusammen. Wie schon gezeigt wurde, gelingt eine fehlerfreie Trennung mit signalangepaßter Filterung. Aufgrund der Überlappung ergibt sich eine hervorragende spektrale Effizienz, die im Vergleich mit Einträgerverfahren fast um den Faktor 2 besser ist, denn mit der Symbolrate r_s erhält man für N Teilkanäle die Gesamtbandbreite

$$B \approx (N+1) \cdot r_S \approx N \cdot r_s \quad \text{bei vielen Teilkanälen} \qquad (5.115)$$

Die spektrale Effizienz wird auch besonders deutlich, wenn man das Summenspektrum in Bild 5.34 rechts oben betrachtet. In der spektralen Darstellung sind FH- und OFDM-Signalformen praktisch identisch. Jedoch wird bei FH stets von unmodulierten Signalformen ausgegangen und die Information ist in der Abfolge der Frequenzen enthalten. Bei OFDM hingegen wird jede Signalform moduliert und überträgt so einen Teil des Datenstroms. Typisch ist dabei eine hohe Anzahl orthogonaler Signalformen, die im Gegensatz zu FH auch nicht nacheinander, sondern parallel gesendet werden. Das Sendesignal ist also die Summe vieler modulierter Trägersignale und bekommt dadurch einen recht komplizierten Zeitverlauf. Aufgrund der hohen Zahl der Träger und der notwendigen parallelen Erzeugung und parallelen empfängerseitigen Verarbeitung ist es bei typischen OFDM-Anwendungen nicht sinnvoll, Korrelatorbänke Bild 5.33Bild 5.33Bild 5.33Bild 5.33zu verwenden. Vielmehr erweist es sich jetzt als vorteilhaft, die Signalsynthese mittels einer inversen diskreten Fouriertransformation (IDFT, IFFT) durchzuführen und für die empfängerseitige Verarbeitung eine DFT bzw. FFT einzusetzen.

Ein Empfangssignalvektor $s(k)$ (auch OFDM-Symbol genannt) besteht aus N Abtastwerten des Empfangssignals im Abtastraster T_A und hat somit die Dauer $T=N\cdot T_A$. Er wird einer DFT unterzogen und liefert das Spektrum $S(n)$ an genau N Stellen, die der Lage der gesendeten Träger entsprechen. Mit $1/T=1/N\cdot T_A=\Delta f$ ergibt sich die korrekte Frequenzauflösung. Die DFT liefert ein komplexes Ergebnis, so daß für jeden Träger die beiden Größen Betrag und Phase zur Verfügung stehen. Damit können die gesendeten Daten zurückgewonnen werden.

Wie schon erwähnt, erhält bei OFDM jeder Träger einen Teil der zu übertragenden Daten. Die Verteilung muß dabei nicht zwangsläufig gleichmäßig sein, und es muß auch nicht überall das gleiche Trägermodulationsverfahren benutzt werden. Das ist ein weiterer Vorteil von OFDM, den man immer dann ausspielen kann, wenn entsprechende Kenntnisse über den Kanal vorliegen. Falls man von bestimmten Teilen des Übertragungsbandes weiß, das dort wenig Dämpfung und Störung vorhanden ist, kann man die dort liegenden Träger mit komplexen Modulationsverfahren hoher spektraler Effizienz wie z.B. vielstufige QAM beaufschlagen. In anderen Teilen des Übertragungsbandes, wo nur geringer Störabstand zu erwarten ist, wird man z.B. BPSK einsetzen. Ganz schlechte oder auch aufgrund von Frequenzzuteilungsvorschriften „verbotene“ Frequenzbereiche können durch Nullsetzen der entsprechenden Spektrallinien vor der IFFT ausgeblendet werden. Es ist wichtig, sich klar

zu machen, daß die Modulation auf die Zusammensetzung des OFDM-Spektrums keinen Einfluß hat. Denn jede Signalform ist während ihrer gesamten Dauer T rein sinusförmig. Das Aussehen des zugehörigen Spektrums wird nur durch T beeinflußt und ist wegen

$$rect\left(\frac{t}{T}\right) \circ\!\!-\!\!\bullet\, T \cdot si(\pi \cdot T \cdot f)$$

sin(x)/x-förmig.

Im unteren Teil von Bild 5.34 sind die Zusammenhänge anhand eines Beispiels, in dem QPSK als Trägermodulation eingesetzt wird, verdeutlicht. Man sieht rechts unten vier Trägersignale gleicher Frequenz und der Dauer T, wobei eine ganze Anzahl von Perioden in T paßt. Sie repräsentieren eine Signalform und belegen im Spektrum somit nur einen Platz. Die Nullphase trägt die Information, d.h. es werden 4 Positionen, nämlich 0°, 90°, 180° und 270° unterschieden. Links ist das in einem Signalraumdiagramm dargestellt. Die Punkte können als Endpunkte von Vektoren interpretiert werden, deren Betrag gleich ist und deren Phase die vier genannten diskreten Werte annehmen kann. Einem Punkt kann deshalb jeweils die Information 2 bit zugeordnet werden und er repräsentiert ein Symbol. Für den Zusammenhang mit den Datenbits könnte folgende Zuordnung getroffen werden:

0° ⇒ 00
90° ⇒ 01
180° ⇒ 10
270° ⇒ 11

Hätte man 16-QAM als Trägermodulation gewählt, dann ergäben sich im Signalraum 16 Punkte, entsprechend 16 Symbolen, von denen jedes 4 bit an Information tragen könnte. Bei der empfängerseitigen Demodulation müßten in diesem Fall 12 diskrete Phasenwerte und 3 Amplitudenstufen unterschieden werden.

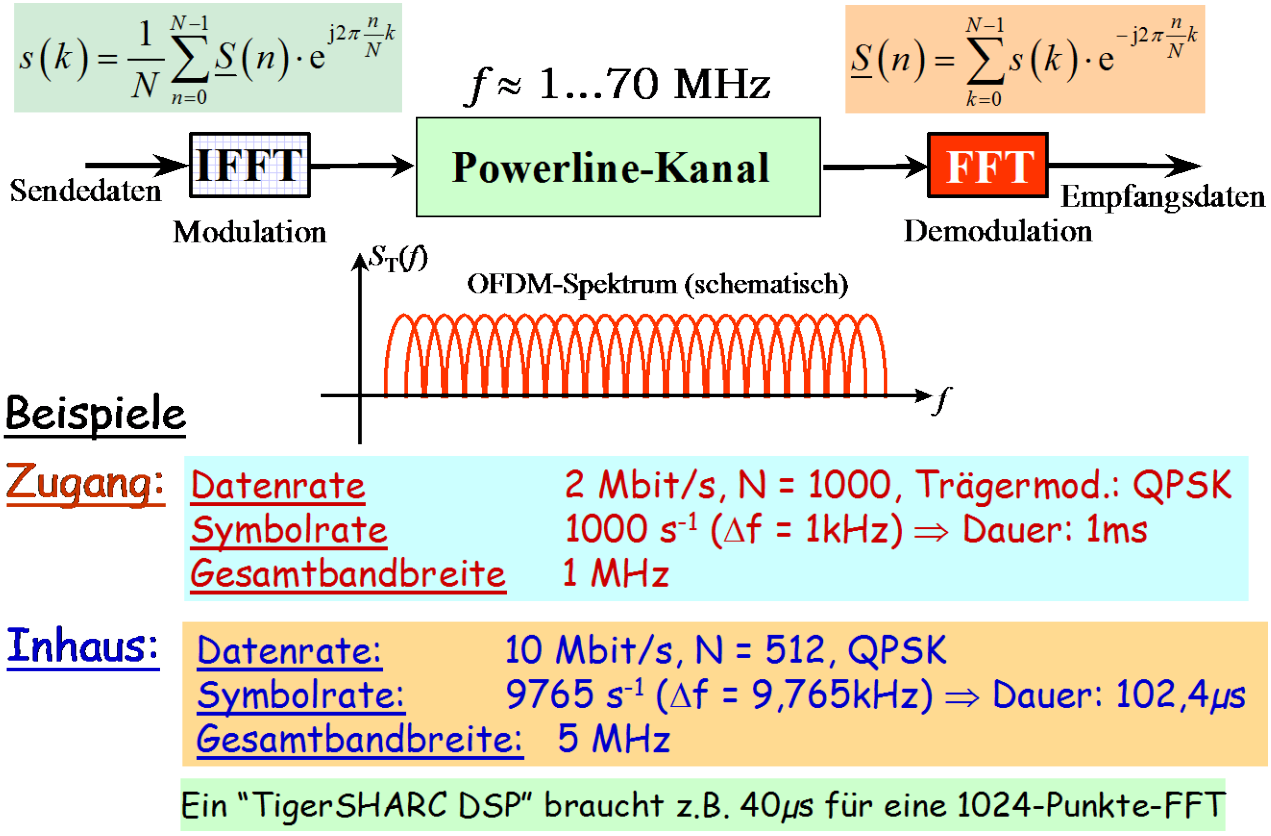


Bild 5.35: Illustration der OFDM-Anwendung an zwei Systembeispielen

An zwei in Bild 5.35 dargestellten Systembeispielen werden die Vorteile von OFDM zusammenfassend verdeutlicht. Ein Powerline-Kommunikationssystem für den Zugangsbereich (letzte Meile) soll eine Datenrate von 2 Mbit/s zur Verfügung stellen, wobei OFDM mit N=1000 Trägern und QPSK als Trägermodulation verwendet wird. Damit hat man die Symbolrate r_s=1 MHz, weil jedes Symbol 2 bit trägt. Jeder der 1000 Träger soll die gleiche Information übertragen. Dann ergibt sich die OFDM-Symboldauer T=1ms. Die Sendedaten werden zunächst zu Symbolen zusammengefaßt (je 2 bit bilden ein Symbol) und jedem dieser Symbole wird ein komplexer Fourierkoeffizient $S(n)$ zugeordnet, so daß sich 1000 derartiger Koeffizienten ergeben. Das Kernstück des Senders kann z.B. ein DSP, der die jetzt benötigte komplexe IFFT ausführt, wofür 1 ms an Verarbeitungszeit zur Verfügung steht. Durch die hohe Trägerzahl ist die Signalformdauer entsprechend lang. Durch weitere Erhöhung der Trägerzahl könnte man entweder die Datenrate vergrößern oder die Signalformdauer weiter verlängern. Letzteres würde die Immunität gegen lange Echos steigern.

Wie groß der Spielraum mit heutiger DSP-Technik ist, läßt sich am Beispiel eines preisgünstigen und ausgereiften Standard-DSPs vom Typ TigerSHARK (Analog Devices) verdeutlichen: Dieser Baustein kann eine komplexe FFT oder IFFT mit N=1024 in 40µs ausführen. D.h. bezüglich dieser Schlüsseloperationen wäre eine Erhöhung der Geschwindigkeitsanforderungen um den Faktor 25 möglich, so daß auch einer DSP-basierten Realisierung des zweiten in Bild 5.35 unten skizzierten Beispiels mit einer Datenrate von 10 Mbit/s bei N=512 Trägern nichts entgegensteht.

5.8.1.1 Grundstruktur eines OFDM-Senders

Bild 5.36 zeigt den Aufbau eines typischen OFDM-Sendesystems. Der von der Datenquelle kommende Bitstrom wird zuerst eine Kanalcodierung unter Einschluß von Interleaving[11] unterworfen. Dann wird der codierte Bitstrom seriell/parallel gewandelt und in N Gruppen zu je ν bit aufgeteilt.

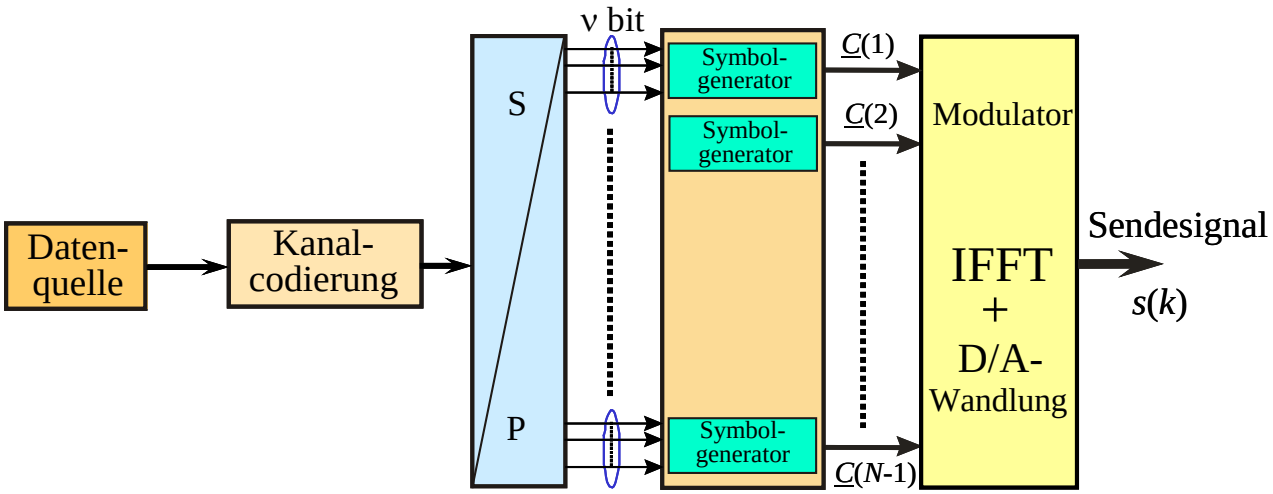


Bild 5.36: Grundaufbau eines OFDM-Senders

Jede der Gruppen repräsentiert ein Symbol, das einem der N Teilkanäle zugeordnet wird. Bei dem ersten OFDM-Beispiel aus Bild 5.35 mit N=1000 und QPSK-Modulation wäre ν=2, so daß immer ein Datenblock von 2000 bit parallel auf die Symbolgeneratoren zu führen wäre. Die Symbolgeneratoren erzeugen jeweils einen komplexen Fourierkoeffizienten $C(n)$, der das zugehörige Symbol im Signalraum abbildet.

Für das QPSK-Beispiel aus Bild 5.34 könnte das Ergebnis bei Normierung der Koeffizienten auf Eins z.B. wie folgt aussehen:

[11] Verwürfelung der Datenbits, damit ein Bündelfehler keine direkt aufeinander folgenden Bits trifft und verfälscht

00 ⇒ 1 10 ⇒ −1

01 ⇒ +j 11 ⇒ −j

Es ist auch möglich, die Abbildung von der Bit- auf die Symbolebene für jeden Teilkanal verschieden durchzuführen, d.h. man könnte für Teilkanäle mit ungünstigem Signalstör-Verhältnis BPSK wählen und für sehr gute Kanäle beispielsweise 16-QAM.

Die komplexen Symbole, die die Spektrallinien darstellen, werden jetzt der IDFT unterzogen und man erhält N Abtastwerte von $s(k)$ als zu sendende Signalform. Diese gelangen der Reihe nach auf einen D/A-Wandler mit nachfolgendem Rekonstruktionsfilter. Darauf folgen Sendeendstufe und Koppeleinrichtung.

5.8.1.2 Grundstruktur eines OFDM-Empfängers

Bild 5.37 zeigt den Übertragungskanal mit der Impulsantwort $h(k)$, einer additiven Störung $n(k)$ und den typischen Aufbau eines OFDM-Empfängers. Das Sendesignal wird auf dem Kanal verzerrt und gestört. Vom Empfangssignal werden jeweils N Abtastwerte im Abtastraster T_A genommen und A/D gewandelt. Da $N \cdot T_A$ gleich der OFDM-Symboldauer T ist, hat man jetzt einen Empfangsvektor $e(k)$ der Läge N, der einer FFT unterzogen wird. $e(k)$ unterscheidet sich vom Sendevektor $s(k)$ durch die Beeinflussung in Amplitude und Phase, die der Kanal mit seiner Übertragungsfunktion ausübt, und durch die überlagerten Störungen.

Nach der FFT hat man einen Ergebnisvektor $E(n)$, wiederum der Länge N, der die komplexen Fourierkoeffizienten des Empfangssignals repräsentiert.

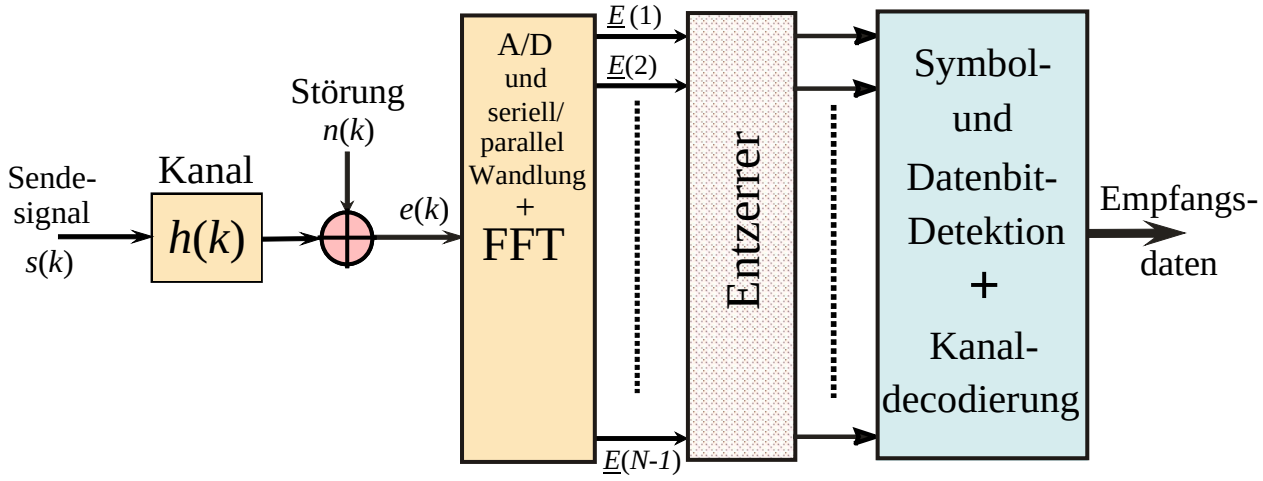


Bild 5.37: Übertragungskanal und OFDM-Empfängeraufbau

In manchen Fällen lohnt es sich, vor der Symbol- und Bitdetektion noch eine Entzerrung vorzunehmen, die den dämpfenden Einfluß der Übertragungsfunktion des Kanals weitgehend eliminieren kann. Eine einfache Möglichkeit dazu wird hier vorgestellt. Wie schon erwähnt, kann bei jedem der OFDM-Teilkanäle Schmalbandigkeit vorausgesetzt werden, so daß Dämpfung und Gruppenlaufzeit jeweils konstant sind. Damit kann jeder Teilkanal durch einen komplexen Übertragungsfaktor

$$H_i = \alpha_i \cdot e^{j\varphi_i}, \text{ mit } i=0...N\text{-}1 \qquad (5.116)$$

beschrieben werden. Während einer „Trainingsphase“ wird eine dem Empfänger bekannte Symbolfolge über den Kanal geschickt. Mit Hilfe dieser Folge ist der Empfänger in der Lage, die Übertragungsfaktoren H_i für jeden Teilkanal zu bestimmen und zu speichern. Eine solche Trainingsphase

ist bei stationären Kanälen nur in sehr großen Zeitabständen erforderlich. In der Praxis genügt oft eine einmalige Trainingssequenz zu Beginn einer Übertragung.

Die Entzerrung erfolgt, wie in Bild 5.38 dargestellt, durch eine Multiplikation der Fourierkoeffizienten $E(n)$ mit den inversen Übertragungsfaktoren aus (5.116), also

$$H_i^{-1} = \frac{1}{H_i} = \frac{1}{\alpha_i} \cdot e^{-j\varphi_i}, \text{ mit } i=0...N\text{-}1. \tag{5.117}$$

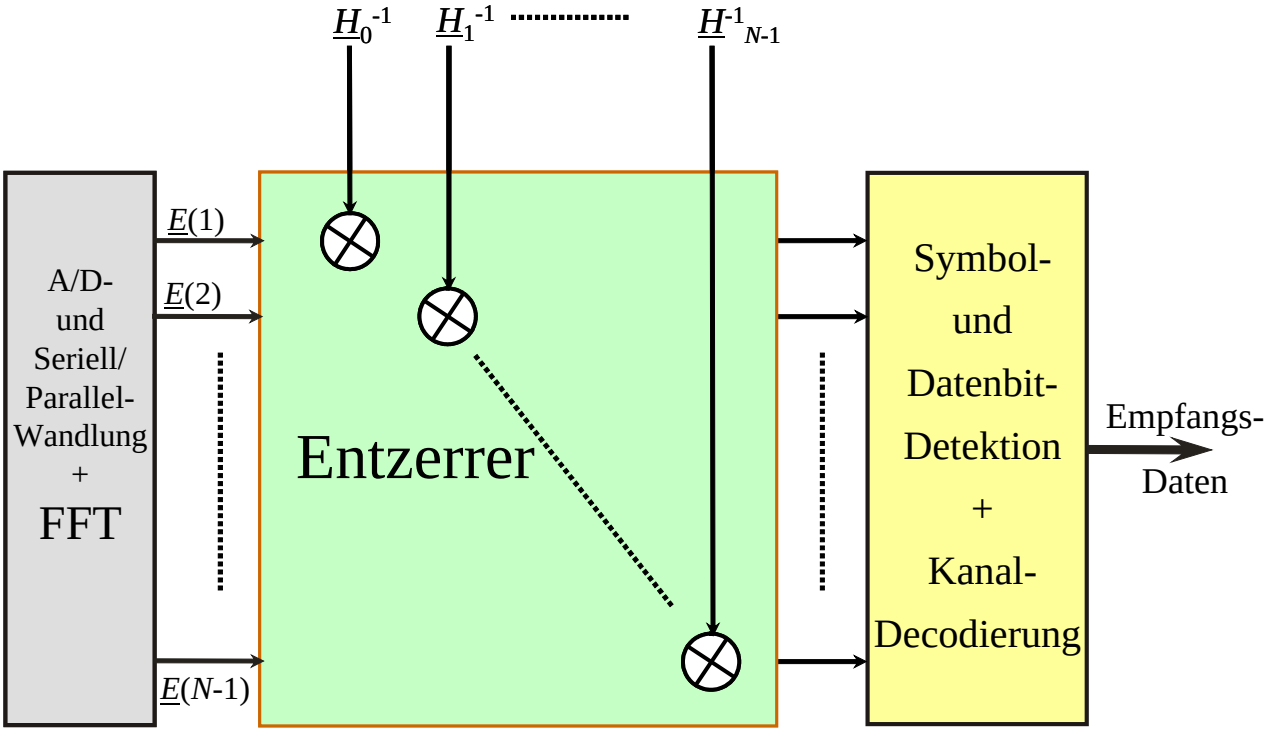


Bild 5.38: Kanalentzerrung bei OFDM

6 Anwendungsspezifische Signalverarbeitungssysteme

6.1 Die Hardware-Beschreibungssprache VHDL

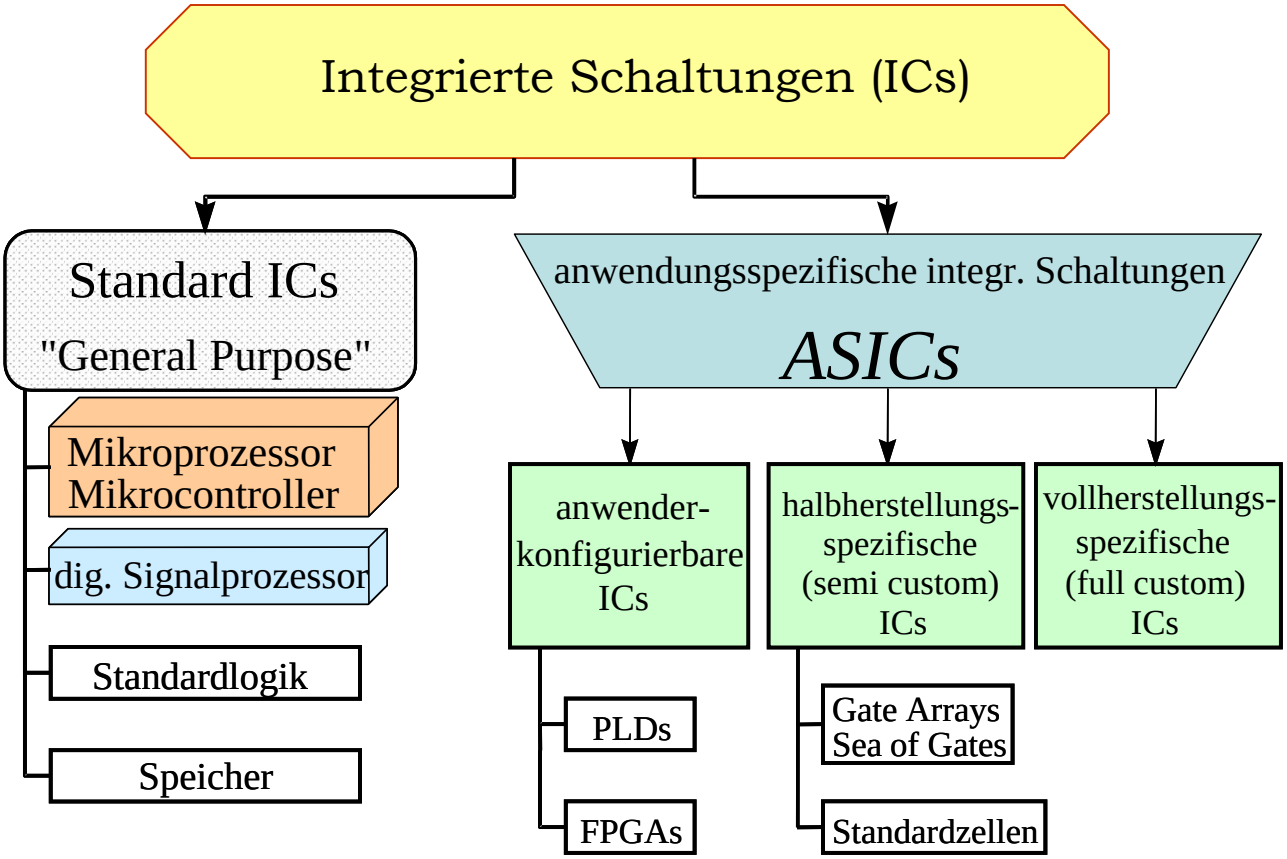
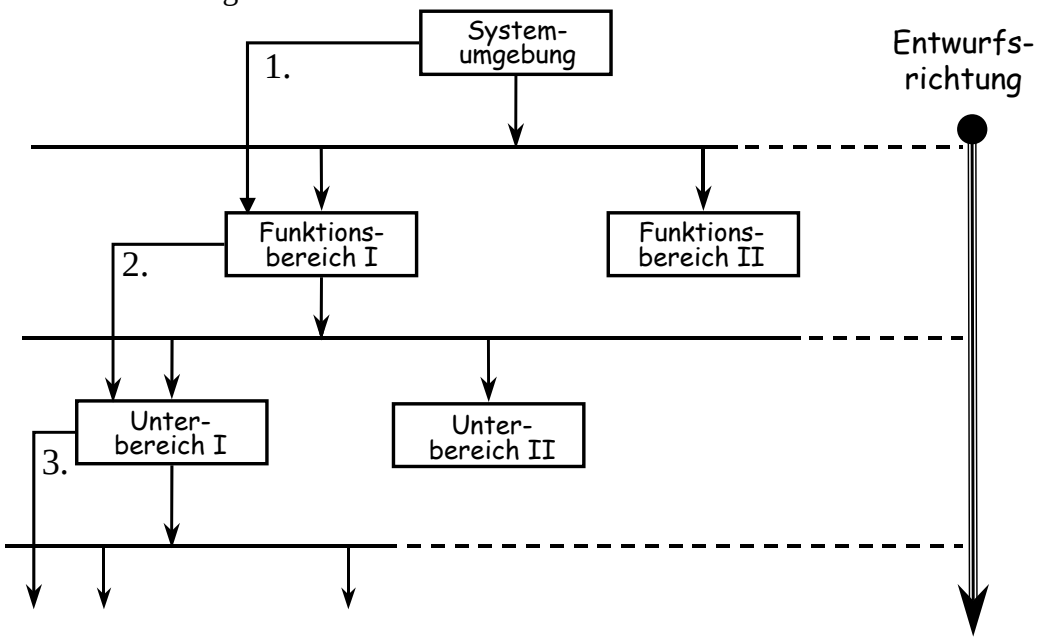


Bild 6.1: Übersicht über die Varianten digitaler integrierter Schaltungen

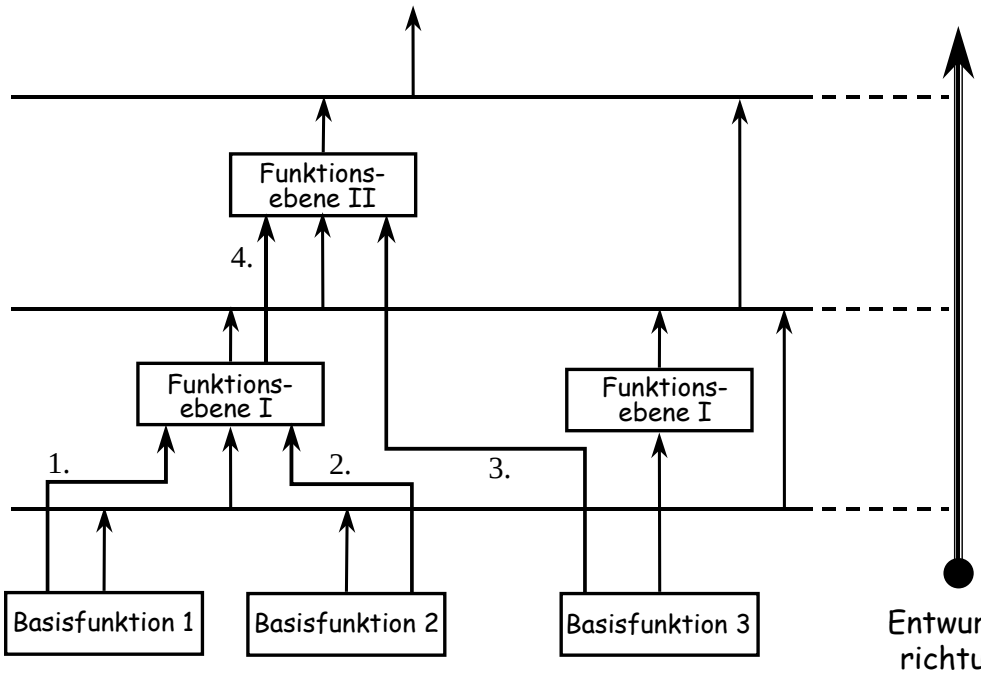
Beim Entwurf digitaler integrierter Schaltungen – wie sie im rechten Teil von Bild 6.1aufgeführt sind – kann man grundsätzlich zwei verschiedene Methoden anwenden.



Für große Projekte, die in Teamarbeit bewältigt werden müssen, empfiehlt sich die in Bild 6.2 skizzierte Top-Down Vorgehensweise. Sie geht von der Systemsicht aus und zerlegt die darunter liegenden Ebenen in die jeweils benötigte Detaildarstellung.

Bild 6.2: Schema des Top-Down-Entwurfs

Es ergibt sich eine klare Gliederung, die das Endziel – das fertige, aus vielen aneinander angepaßten Einzelteilen bestehende System – stets im Auge behält. Eine Verzettelung in Details wird bei der Top-Down-Methode vermieden.



Obwohl der in Bild 6.3 gezeigte Bottom-Up Entwurfsstil diese Gefahr birgt, hat er dennoch seine Berechtigung und kann immer dann erfolgreich praktiziert werden, wenn bereits Vorerfahrung in Form von realisierten Basisfunktionsblöcken vorhanden ist.

Bild 6.3: Schema des Bottom-Up-Entwurfs

Solche Blöcke können schnell zu größeren übergeordneten Funktionseinheiten kombiniert werden, die dann auf den höheren Ebenen getestet werden können, ohne daß jede Basisfunktion nochmals verifiziert werden muß. Das Ziel – ein komplettes funktionsfähiges System – kann mit der Bottom-Up Methode vom erfahrenen Entwickler (oder Entwicklungsteam) oft schneller erreicht werden, als mit der Top-Down Methode. Allerdings darf nicht übersehen werden, daß insbesondere bei großen Projekten eine Bottom-Up Vorgehensweise deutlich höherer Anforderung an das Projektmanagement stellt.

Elektronische Systeme aller Art – nicht nur digitale integrierte Schaltungen – sind seit den 80-er Jahren derart komplex geworden, daß eine Übersicht anhand von Standard-Dokumenten (Handbücher, Betriebs- und Reparaturanleitungen) nicht mehr zu gewinnen ist. Vor allem ein Vergleich zwischen verschiedenen Herstellern ist – nicht zuletzt durch die verschiedenartigsten Schutzmaßnahmen der Hersteller für ihr Know-How und ihr geistiges Eigentum (Intellectual Property ≡ IP) – praktisch unmöglich geworden. Aus dieser unbefriedigenden Situation, die natürlich nicht nur den zivilen Handel und die Industrie betraf, sondern auch Behörden und vor allem das Militär, entstand Anfang der 80-er Jahre in den USA eine Initiative, die die Grundlage für die heute weltweit am weitesten verbreitete Hardwarebeschreibungssprache VHDL bildete.

6.1.1 Historische Entwicklung von VHDL

Die Anfänge von VHDL[12] reichen bis in die frühen achtziger Jahre zurück. Im Rahmen des **VHSIC**[13]-Programms wurde in den USA vom Verteidigungsministerium (Department of Defense, **DoD**) nach einer Sprache zur Dokumentation elektronischer Systeme gesucht. Dadurch sollten die enormen Kosten für **Wartung und Nachbesserung** bei militärischen Systemen, die etwa die Hälfte der Gesamtkosten ausmachten, reduziert werden. Von besonderer Wichtigkeit war die klare Beschreibung von komplexen Schaltungen sowie die **Austauschbarkeit** von Modellen zwischen verschiedenen Entwicklungsgruppen. Das **VHSIC**-Programm startete mit einem Workshop zur Festle-

[12] **V**HSIC **H**ardware **D**escription **L**anguage
[13] **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit

gung der Ziele. Im Juli 1983 bekamen die Firmen Intermetrics, IBM und Texas Instruments den Auftrag, die neue Sprache zu entwickeln. Sie sollte sich an die im militärischen Bereich weit verbreitete Programmiersprache **ADA** anlehnen. Im August 1985 konnte eine erste Version - VHDL V7.2 - vorgestellt werden, die dann im Februar 1986 zur Standardisierung an **IEEE**[14] übergeben wurde. Unter Beteiligung zahlreicher Software- und Hardwarehersteller aus der **CAE**[15]-Industrie gewann VHDL zunehmend Anhänger und wurde schließlich im Dezember 1987 als **IEEE l076-1987** zum ersten und bislang einzigen Standard für Hardwarebeschreibungssprachen. Der Standard definiert allerdings nur die **Syntax** und **Semantik** der Sprache selbst, nicht jedoch ihre Anwendung bzw. ein einheitliches Vorgehen bei der Anwendung.

Seit September 1988 müssen alle Elektronik-Zulieferer des **DoD** VHDL-Beschreibungen ihrer Komponenten und Subkomponenten bereitstellen. Ebenso sind VHDL-Modelle von Testumgebungen mitzuliefern.

Nach den **IEEE**-Richtlinien muß ein Standard alle **fünf Jahre** überarbeitet werden, um nicht zu verfallen. Aus diesem Grund wurde im Zeitraum 1992-1993 die Version **IEEE 1076-1993** definiert. Die Dokumentation des neuen Standards, im 'Language Reference Manual' (**LRM**) erschien Mitte 1994. Seit Beginn der neunziger Jahre hat sich VHDL weltweit etabliert. Der **1076-l993**-Standard entstand bereits weitgehend losgelöst vom **DoD** unter internationaler Beteiligung. Europa wirkte unter anderem über **ESPRIT** daran mit. Auch Asien - vor allem Japan - hat VHDL akzeptiert. Konkurrenz besteht heute praktisch nur noch durch '**Verilog HDL'** vor allem in den USA, weil der führende **CAE**-Hersteller Cadence diese Sprache in seine Werkzeuge eingebaut hat. Während **Verilog** über lange Zeit als 'Cadence-proprietär' anzusehen war, ist auch diese HDL mittlerweile ein offener Standard, der in den meisten **CAE**-Werkzeugen für den Entwurf integrierter Schaltungen gleichberechtigt neben VHDL zur Verfügung steht.

6.1.2 Entwurfsebenen und „Sichtweisen" integrierter Schaltungen

Sobald man sich mit dem Entwurf integrierter Schaltungen befaßt, wird man zwangsläufig mit einem Modell konfrontiert, das die verschiedenen Entwurfsebenen aus den drei Sichten Verhalten, Struktur und Geometrie wiedergibt. Die Darstellung ist unter der Bezeichnung Gajski-Walker-Modell bekannt. Die einzelnen Ebenen werden im folgenden kurz aus den drei Sichtweisen betrachtet, und ihre Rolle beim Entwurf digitaler integrierter Schaltungen wird erläutert.

- Systemebene

Die Systemebene beschreibt die grundlegenden Charakteristika einer Schaltung. In der Beschreibung werden typische Blöcke, wie Speicher, Prozessoren oder Interface-Einheiten verwendet, die jeweils durch ihre Funktionalität (z.B. Befehlssatz, Protokoll) charakterisiert werden. Auf dieser Ebene dienen „natürliche" Sprache und Skizzen als Beschreibungsmittel. Die Systemebene liefert noch keine Aussagen über ***Signale***, deren Zeitverhalten und weitere funktionale Details. Vielmehr dient sie einer groben Aufteilung der gesamten Schaltungsfunktion. Aus geometrischer Sicht beinhaltet die Systemebene beispielsweise eine rudimentäre Aufteilung der Chipfläche.

[14] **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
[15] **C**omputer **A**ided **E**ngineering

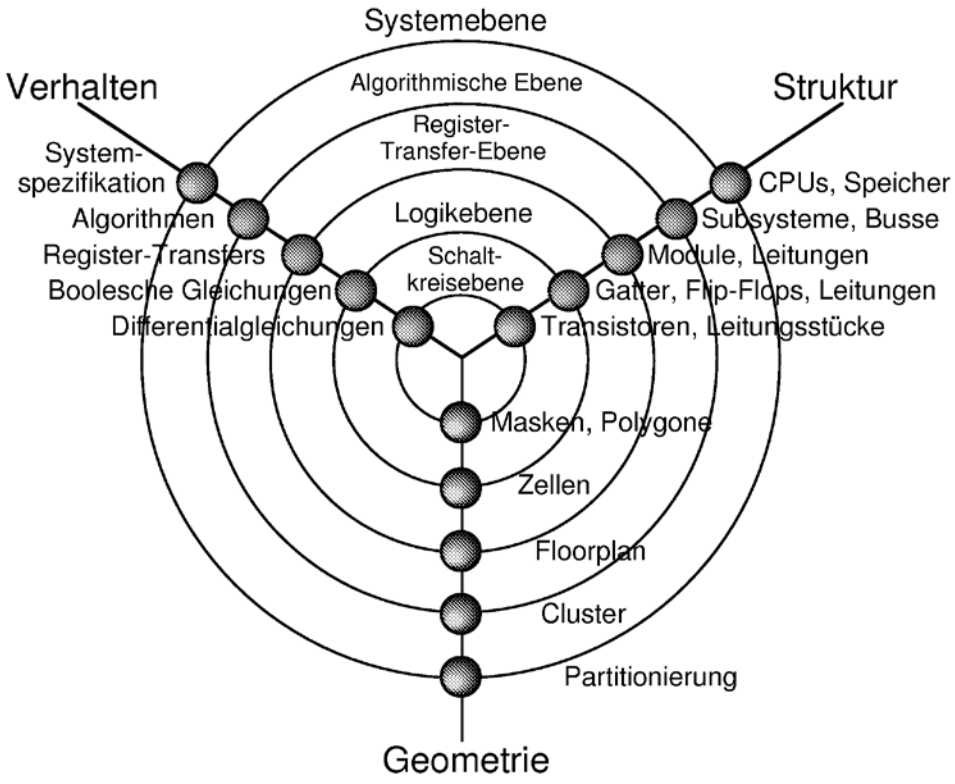


Bild 6.4: Das Gajski-Walker-Modell (auch Y-Modell genannt)

- Algorithmische Ebene

Hier kann eine Schaltungsbeschreibung durch nebenläufige (d.h. parallel ablaufende) Algorithmen erfolgen. Typische Elemente sind dabei Funktionen, Prozeduren, Prozesse und Kontrollstrukturen.

Aus strukturaler Sicht wird hier ein elektronisches System durch Blöcke beschrieben, die über Signale miteinander kommunizieren.

Die Verhaltenssicht enthält z.B. eine algorithmische Darstellung mit Variablen und Operatoren. Es wird noch kein Bezug zur Struktur der späteren Realisierung hergestellt. So fehlen z.B. auch noch alle zeitlichen Vorgaben bezüglich Takt- oder Rücksetzsignalen.

- Register-Transfer-Ebene

Auf der Register-Transfer-Ebene (engl.: **R**egister **T**ransfer **L**evel, **RTL**) werden die Eigenschaften einer Schaltung z.B. durch mathematische oder logische Operationen und durch die Transfermöglichkeiten der zu verarbeitenden Daten zwischen Registern spezifiziert.

In die Beschreibung fließen in der Regel bereits Takt- und Rücksetzsignale ein. Die einzelnen Operationen können dann den Zuständen oder Flanken dieser Signale zugeordnet werden, so daß zeitliche Eigenschaften schon weitgehend definiert sind. Aus strukturaler Sicht werden Elemente wie Register, Zähler, Adreßdecoder, Multiplexer oder Addierer durch Signale miteinander verknüpft.

Aus Verhaltenssicht ergeben sich meist Beschreibungen in Form endlicher Automaten. In geometrischer Sicht wird die Grobeinteilung der Chipfläche zu einem ’Floorplan’ verfeinert.

Es gibt eine Reihe von CAE–Werkzeugen für die automatische Umsetzung einer Verhaltensschreibung in eine strukturale Beschreibung auf der Logikebene, d.h. RTL-Beschreibungen sind in der Regel synthetisierbar.

- Logikebene

Hier werden die Eigenschaften eines elektronischen Systems durch logische Verknüpfungen und deren zeitliche Eigenschaften (Verzögerungszeiten) beschrieben.

Der Verlauf der Ausgangssignale ergibt sich dabei durch die Anwendung dieser Verknüpfungen auf die Eingangssignale. Die Signalverläufe sind wertdiskret, z.B. '**low**', '**high**', '**tristate**'.

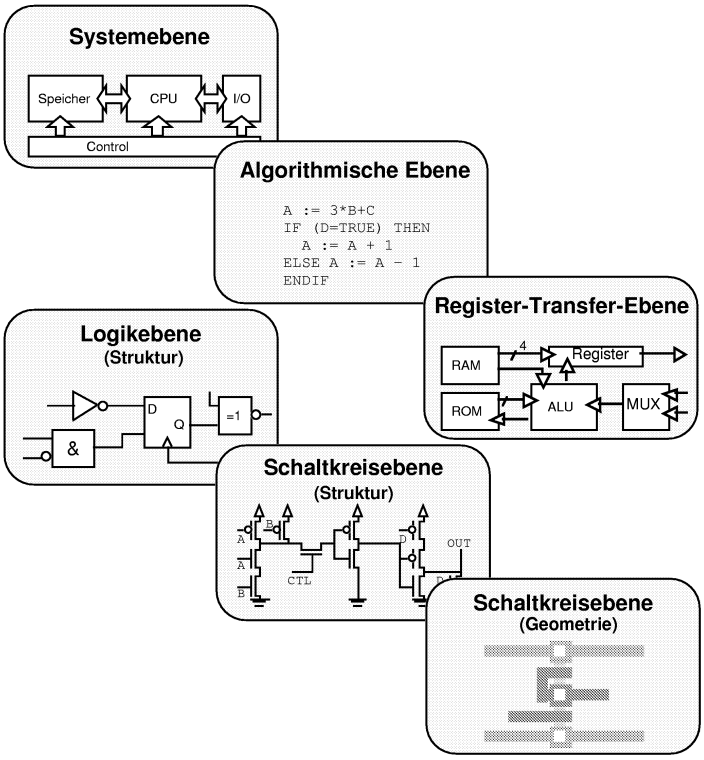


Bild 6.5: Anschauliche Übersicht über die Entwurfsebenen

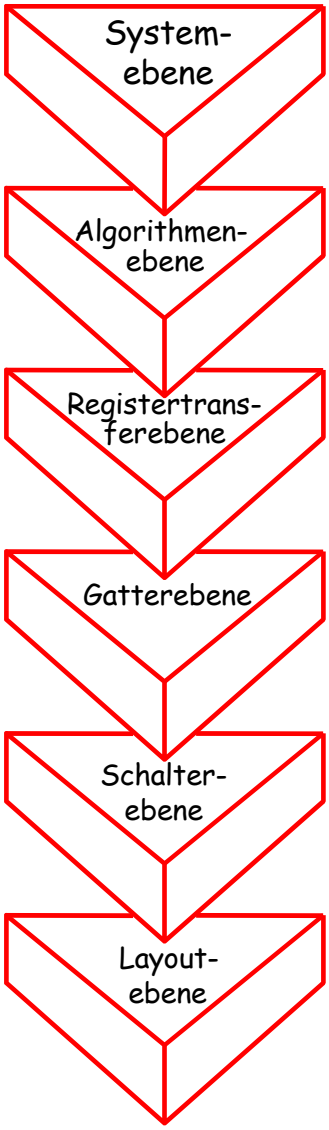


Bild 6.6: Symbolische Darstellung der Entwurfsebenen

Aus strukturaler Sicht wird ein Design durch eine Zusammenschaltung von Grundelementen (z.B. AND-, OR-, EXOR–Gatter, Flip-Flops) dargestellt. Diese Grundelemente werden dabei von einer Bibliothek zur Verfügung gestellt. Innerhalb dieser Bibliothek sind die Eigenschaften der Grundelemente definiert. Hier sind die Charakteristika der einzelnen Zellen der Zieltechnologie in vereinfachter Form abgelegt, z.B. 190nm-Standardzellen-Prozeß der Fa. ATMEL, FPGA vom Typ ACEX

EP 1K 100 Fa. ALTERA. Zur Erstellung der strukturalen Beschreibung (Gatternetzliste) werden meistens graphische Eingabewerkzeuge verwendet (Schematic Entry).

Aus Verhaltenssicht erfolgt vornehmlich eine Darstellung durch Boolesche Gleichungen, z.B. Y=(A AND B) XOR C oder durch Werte- bzw. Funktionstabellen. Der Übergang von der Verhaltenssicht auf die strukturale und geometrische Sicht erfolgt mit Hilfe von Synthesesoftware.

- Schaltkreisebene

Hier liegt struktural eine „verfeinerte“ Netzliste vor, d.h. es sind keine logischen Gatter oder noch komplexere Funktionseinheiten zusammengeschaltet, sondern Bauelemente wie Transistoren, Kondensatoren und Widerstände.

Einzelne Module werden nicht mehr durch logische Funktionen mit einfachen Verzögerungen beschrieben, sondern im Detail durch ihren physikalischen Aufbau.

Aus geometrischer Sicht hat man Polygonzüge vor sich, die z.B. unterschiedliche Dotierungsbereiche auf der Halbleiteroberfläche definieren.

Die Verhaltenssicht benötigt vornehmlich Differentialgleichungen zur Schaltungsbeschreibung, so daß Simulationen auf dieser Ebene entsprechend aufwendig und rechenintensiv sind.

Signale auf Schaltkreisebene können im Gegensatz zur Logikebene beliebige Werte zu beliebigen Zeitpunkten annehmen, d.h. sie sind zeit- und wertkontinuierlich.

Jede der vorgestellten Entwurfsebenen erfüllt einen bestimmten Zweck. Während auf den oberen Ebenen hohe Komplexitäten beherrschbar sind, bieten die unteren Ebenen Details und Genauigkeit. Systemebene und algorithmische Ebene eignen sich daher für die Dokumentation und Simulation eines Gesamtsystems, während die Register-Transfer-Ebene für die Blocksimulation und synthesegerechte Modellierung geeignet ist. Bild 6.5vermittelt einen anschaulichen Eindruck der Beschreibungen auf den verschiedenen Ebenen, während Bild 6.6 schematisch die hierarchische Struktur hervorhebt.

Auf jeder Ebene wird nur die benötigte Genauigkeit geboten; unwichtige Details sind nicht sichtbar. Eine eindeutige Einordnung einer Beschreibung in eine bestimmten Ebene gelingt nicht immer; es gibt somit auch ’Multi-Level’-Beschreibungen.

6.1.3 Der Nutzen von Hardwarebeschreibungssprachen

In den Anwendungen digitaler Technik trifft man Prozessoren (MP, MC oder DSP) immer seltener als diskrete Bauelemente an. Mehr und mehr sind Prozessoren heute in Systeme integriert (oder eingebettet, engl.: embedded), die in der Regel auch analoge Bestandteile haben. Oft ist dabei die Komplexität – gemessen an der Zahl der Transistoren oder der benötigten Siliziumfläche - der den Prozessor umgebenden Schaltung weitaus größer die des Prozessors selbst. Derartige Systeme sind keineswegs etwas neues, sondern werden seit Jahrzehnten auf Leiterplatten aufgebaut. Aufgrund der enormen Fortschritte der Halbleitertechnologie in den letzten Jahren ist es jedoch heute möglich, neben anwendungsspezifischen Schaltungsteilen auf einem Stück Silizium auch mehrere Standard-Prozessorkerne zu implementieren. Ein „Embedded System“ kann darüber hinaus auch analoge Schaltungsteile einschließen, wie z.B. Analog/Digital- und Digital/Analogwandler oder Operationsverstärker, mit denen analoge Filter oder digital einstellbare Verstärker realisiert sind. Letztere spielen sowohl in der Meß- und Automatisierungstechnik als auch in der Kommunikationstechnik eine wichtige Rolle, d.h. immer dort, wo man es mit Signalen, die einen sehr großen Dynamikbereich umfassen, zu tun hat. Im Zusammenspiel mit den digitalen Schaltungsteilen lassen sich auch „intelligente“, adaptive Konzepte realisieren.

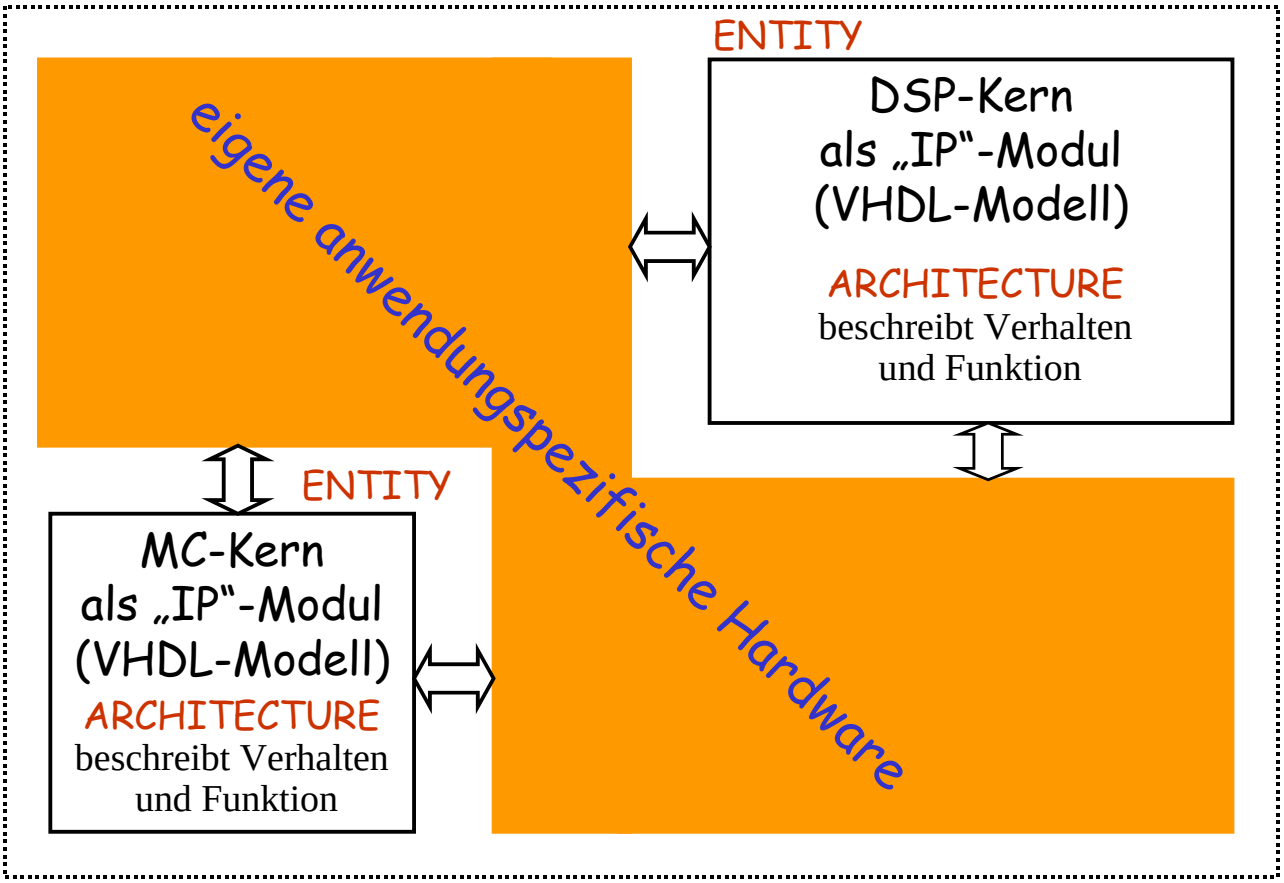


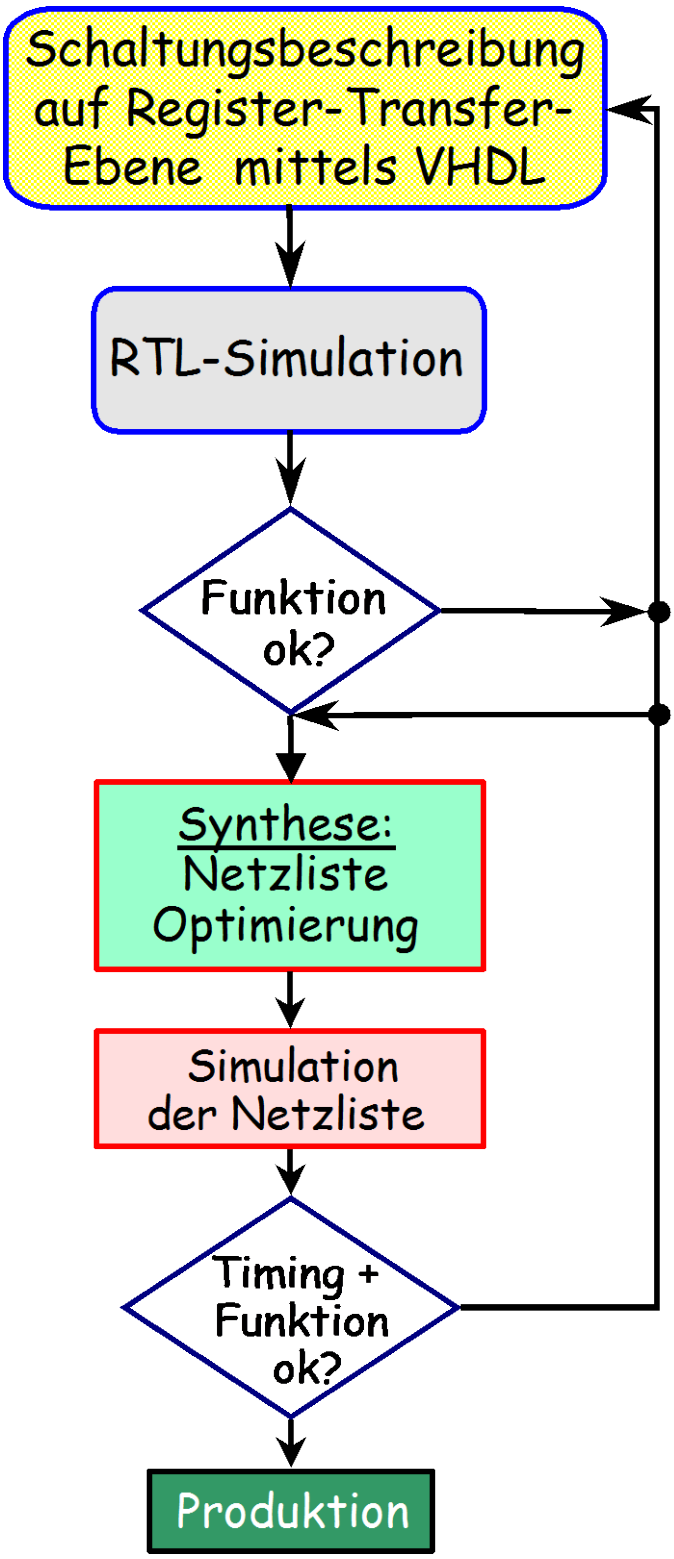
Bild 6.7: Konzept zum Aufbau komplexer Systeme auf Silizium

Die Vorteile, die sich durch eingebettete Systeme ergeben, sind so gewaltig, daß kein Unternehmen sie heute mehr außer acht lassen kann. Auch mit noch so kostengünstiger Leiterplattentechnik gelingt es nicht mehr, Produkte zu schaffen die gegenüber einer monolithischen Integration konkurrenzfähig wären. Bild 6.7 illustriert den Grundgedanken eingebetteter Systeme auf Silizium (**S**ystem **o**n **S**ilicon ≡ SoS).

6.1.4 Einsatz von VHDL beim Entwurf digitaler integrierter Schaltungen

Wie bereits angedeutet, können nicht alle VHDL–Modelle in Schaltungen umgesetzt werden. Ein sicherer Ausgangspunkt ist jedoch immer eine Beschreibung auf der Register-Transfer-Ebene. Schaltungsbeschreibungen auf dieser Ebene sind dem Ingenieur vertraut. Die Umsetzung in den VHDL–Text erfordert allerdings einige Einarbeitung, nicht nur zum Erlernen der Sprachsyntax sondern auch zum Entwickeln eines eigenen Modellierungsstils. Dies ist unbedingt nötig, da VHDL keine Stilvorgaben macht, d.h. ein Sachverhalt kann auf viele verschiede Arten beschrieben werden, die im Ergebnis natürlich auch zu verschieden aufgebauten Schaltungen führen. Da die Eigenschaften der Ergebnisse sich signifikant hinsichtlich Aufwand und Geschwindigkeit unterscheiden können, erhebt sich für den Ingenieur immer die Frage der Optimierung. Dies ist beim Einsatz von VHDL für den Anfänger eine komplexe und schwer überschaubare Angelegenheit, d.h. nur durch Sammeln von Erfahrung im Rahmen zahlreicher Entwürfe kann die nötige Sicherheit erworben werden.

Bild 6.9 gibt einen allgemeinen Überblick über die Schritte vom VHDL–Text bis hin zur Herstellung einer integrierten Schaltung.



An erster Stelle steht nach Fertigstellung der Texteingabe die VHDL–Simulation. Hier wird – noch ganz unabhängig von der Realisierungstechnologie - die Funktion der Schaltungsidee überprüft. Auftretende Fehler und/oder nötige Verbesserungen und Erweiterungen führen solange zurück zum VHDL–Texteditor, bis ein zufrieden stellendes Ergebnis erreicht ist. Dann wird der nächste Schritt, die ***Synthese*** eingeleitet, mit der man die Ebene der Technologieunabhängigkeit verläßt - vgl. auch Bild 6.9. Es entsteht dabei die ***Netzliste***, ein Schaltplan, der aus „logischen Primitiven“ aufgebaut ist, d.h. z.B. aus Elementargattern und Flip-Flops. Die Netzliste wird je nach Synthesewerkzeug noch optimiert, bevor eine Simulation der Schaltung erfolgt. Der wesentliche Unterschied zur VHDL–Simulation ist, daß jetzt das Zeitverhalten der Elemente – so wie es der Hersteller spezifiziert hat – mit einbezogen wird, d.h. typische Verzögerungszeiten von Ein- zu Ausgängen sowie Setup- und Hold-Zeiten. Entdeckte Fehler oder nötige Verbesserungen auf dieser Ebene erfordern Verzweigungen zurück auf den VHDL–Text oder zum Beginn des Syntheseschritts, wo ggf. Vorgabeparameter für die Synthese angepaßt werden müssen.

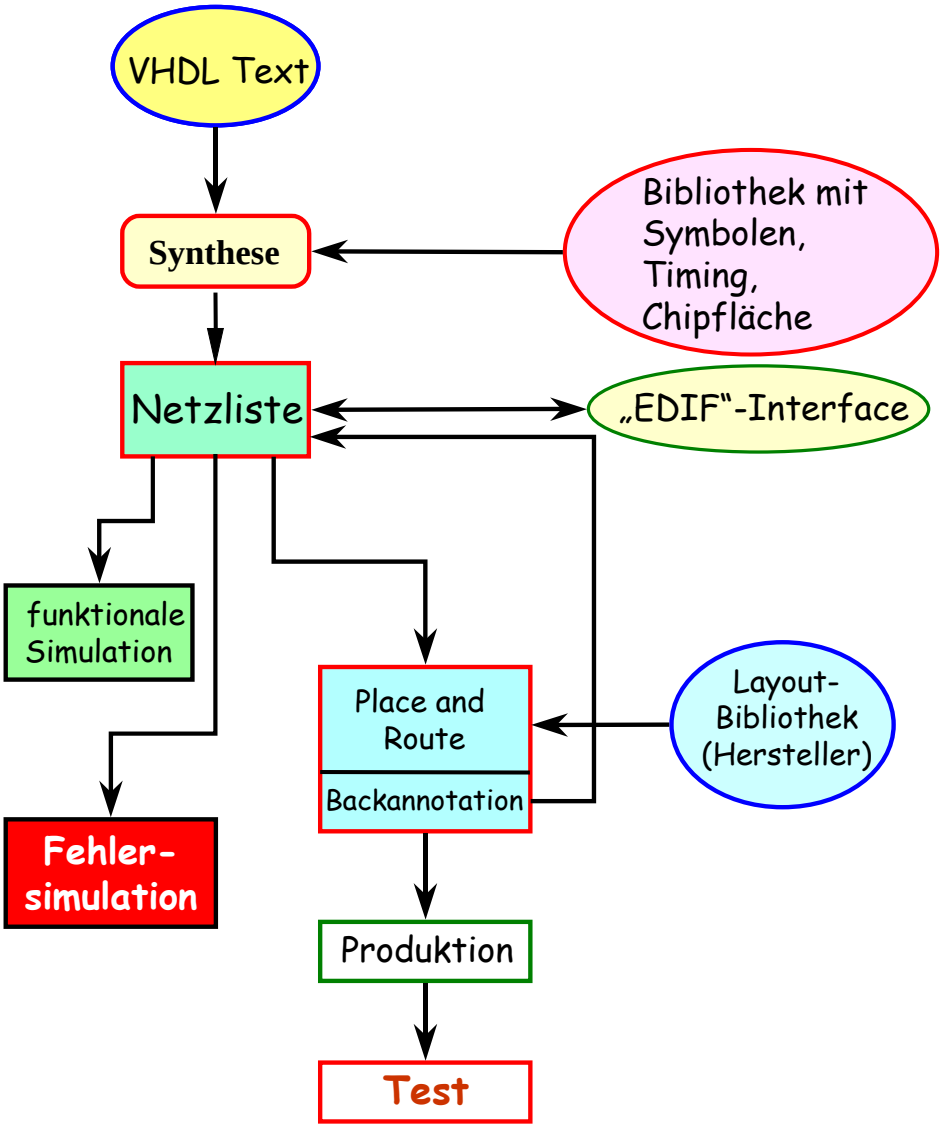
In Bild 6.9 sind die bislang anhand von Bild 6.8 skizzierten Schritte ab der Synthese detaillierter und mit Bezug zur ***Technologie*** dargestellt. Für die Synthese muß demnach eine ***Bibliothek des Herstellers*** zur Verfügung stehen, in der – ähnlich wie in einem Datenbuch – die Symbole der verwendbaren Elemente (z.B. Gatter, Transfergates, Flip-Flops) enthalten sind.

Bild 6.8: Prinzipieller Ablauf von der RTL-Beschreibung bis zur Produktion

Dazu gehören auch ihre Eigenschaften, wie z.B. Signalverzögerung vom Eingang zum Ausgang, Setup- und Hold-Zeiten) Abhängig von der Zieltechnologie (Full-Custom IC, Gate-Array, Cell-Array oder FPGA)[16], enthält die Bibliothek auch Angaben über den Ressourcenverbrauch wie z.B. Chipfläche oder Anzahl der Logikzellen bei FPGAs. Die mit der Synthese entstehende Netzliste

[16] diese ASIC-Varianten werden später noch im Detail analysiert

enthält demnach bereits grundlegende Information über das Zeitverhalten des Entwurfs. Was aber noch fehlt, ist der Einfluß der Verbindungsleitungen.



Die Netzliste ist ein Schaltplan, in dem zwar alle vorkommenden Bauelemente mit ihren Eigenschaften spezifiziert sind, aber noch keine Layoutinformation. Diese wird erst im nächsten Schritt – Place and Route – gewonnen. Es ist trotzdem sinnvoll, mit der Netzliste eine funktionale Simulation durchzuführen. Damit können über die in Bild 6.8 angesprochene RTL-Simulation hinaus zwei wichtige Aspekte verifiziert werden:

- Ist die Übersetzung des VHDL–Textes in die Schaltung fehlerfrei erfolgt?
- Hat das Zeitverhalten der Bauelemente keinen nachteiligen Einfluß auf die Funktion?

Bild 6.9: Vom VHDL-Text zum IC unter Einbezug der Technologie

Wie Bild 6.9 des weiteren zeigt, gibt es zur Netzliste ein EDIF[17]-Interface. Es handelt sich hierbei um ein sehr leistungsfähiges Standardformat, das von den EDA-Werkzeugen aller führenden Hersteller erzeugt und gelesen werden kann. EDIF schließt alle drei Sichten auf allen Ebenen des Y-Modells nach Bild 6.4 ein und wurde schon 1989 vom IEEE zum Industriestandard erhoben.

Je nach Technologie können für den durchgehenden Entwurf – wie er in Bild 6.9 – dargestellt ist, nicht alle benötigen Werkzeuge in einer geschlossenen Umgebung zur Verfügung gestellt werden. Das trifft z.B. auf das ***Placement und Routing*** für Full-Custom ICs oder komplexe Standardzellenentwürfe zu. Diese Aufgabe ist so komplex, daß oft eine Auslagerung auf sehr leistungsfähigere Rechner als PCs – wie z.B. Workstations - erfolgen muß. Dazu kann das EDIF–Format verwendet werden.

Für das Placement und Routing sind – wie in Bild 6.9 angedeutet – weitergehende Bibliotheksinformationen des Herstellers bezüglich der Zieltechnologie erforderlich. Bei Full-Custom ICs oder Standardzellen muß neben der Geometrie der einzelnen Elemente auf der Siliziumoberfläche auch

[17] **E**lectronic **D**esign **I**nterchange **F**ormat

ein Satz von Entwurfsregeln (Design Rules) zur Verfügung gestellt werden. Denn auf dem Silizium sind - ähnlich wie auf einer Leiterplatte - Abstandregeln einzuhalten, die Toleranzen bei der Herstellung berücksichtigen sowie Vorgaben über die Mindestbreite von Leitungen.

Wenn ein FPGA als Zieltechnologie gewählt wird, kann in der Regel in einer geschlossenen PC-basierten Entwurfsumgebung gearbeitet werden. Da bei FPGAs die Zahl der Freiheitsgrade bei Plazierung und Verdrahtung drastisch eingeschränkt ist, genügt die Rechenleistung von PCs. Alle führenden FPGA–Hersteller liefern mittlerweile kostengünstige und leistungsfähige PC-basierte Entwurfsumgebungen, mit denen alle Schritte vom VHDL–Text bis zum Test der fertig konfigurierten Hardware durchgeführt werden können. Als Beispiel wird die von der Fa. ALTERA bereitgestellte Software ***QUARTUS II*** betrachtet.

Nach dem Plazieren und Verdrahten geht der Entwurf – wie man in Bild 6.9 sieht – noch nicht in Produktion, sondern der wichtige Schritt ’Backannotation’ wird vorher ausgeführt. Hierbei wird aus der fertig verdrahteten Schaltung die Netzliste zurück gewonnen, wobei die Verdrahtungsinformation mit einfließt. Das wichtigste sind dabei die zusätzlich entstandenen Verzögerungen. Da bei der heutigen CMOS–Technologie die Schaltgeschwindigkeit der Transistoren bereits bis in den Bereich von Picosekunden vorgedrungen ist, wird der Hauptteil der Verzögerungen durch die Verdrahtung verursacht. Es ist deshalb unabdingbar, die Funktion der Schaltungen nach der Verdrahtung nochmals zu verifizieren. Erst wenn hierbei keine Fehler auftreten, kann die Produktion gestartet werden, die zunächst Muster für den physikalischen Test in der realen Anwendungsumgebung bereitstellt. Die Unterschiede – und damit auch die anfallenden Kosten – sind bei der Musterherstellung für die verschiedenen ASIC-Arten naturgemäß groß. Beim FPGA genügt – wie später noch näher beschrieben wird - ein ’Download’ z.B. über die Druckerschnittstelle eines PC, während bei einem Full-Custom IC oder einer Standardzelle jeweils ein vollständiger Maskensatz für die Halbleiterdotierung und die Metallisierung erforderlich ist. Während die beim FPGA benötigte Schnittstellen-Hardware für rund 5 € hergestellt werden kann, kostet ein Maskensatz zwischen 50.000 ... 250.000 €. Zu den hohen Maskenkosten beim Full-Custom IC oder einer Standardzelle kommt, daß Fehler nicht reparierbar sind oft die Herstellung eines kompletten neuen Maskensatzes erfordern.

FPGAs hingegen sind je nach Technologie sehr oder gar beliebig oft re-programmierbar, so daß Fehler in Sekundenschnelle und praktisch ohne Kosten repariert werden können. Nicht zuletzt dieser Aspekt hat in den letzten Jahren zu einem enormen Wachstum FPGA–basierter Schaltungsentwürfe geführt, und es ist davon auszugehen, daß dieser Trend sich mit noch steigender Tendenz fortsetzt. Für FPGAs spricht auch noch der Wegfall der in Bild 6.9 eingetragenen Fehlersimulation. Diese ist für alle ASICs, die die Mitwirkung des Halbleiterherstellers erfordern, unabdingbar. Denn es geht darum, einen Satz von Testvektoren zu generieren, die an Eingangspins der Schaltung anzulegen sind, während die Reaktionen der Schaltung auf diese - auch Stimuli genannten Signale - mit einer Sollwerttabelle verglichen werden, die aus einer „Gutsimulation“, d.h. der Simulation einer fehlerfreien Schaltung stammt. Es ist wichtig, sich darüber im Klaren zu sein, daß an hier nicht die Funktion der Schaltung geprüft wird, die ja im Verlauf des Entwurfs schon mehrfach simulativ verifiziert wurde, sondern es geht darum, Fehler aller Art aufzudecken, die bei der Herstellung auftreten können. Dazu zählen defekte Transistoren durch Kristallfehler oder fehlerhafte Dotierung (z.B. durch eine verunreinigte Maske), Leitungsunterbrechungen oder Kurzschlüsse sowie fehlerhafte Durchkontaktierungen zwischen den Metallisierungsebenen.

Da jede einzelne Schaltung aus der Produktion dem Test unterzogen werden muß, ist die Zahl der Testvektoren begrenzt – der Testdurchlauf pro Chip sollte im Millisekundenbereich liegen.

Auf den ersten Blick scheint die Aufgabe praktisch unlösbar, einerseits aufgrund der hohen Anzahl von Fehlerquellen und anderseits aufgrund der Tatsache, daß viele interne Knoten der Schaltung von außen, d.h. über die Pins nicht direkt erreichbar sind. In langjähriger Praxis hat sich jedoch ge-

zeigt, daß mit einem sehr einfachen Fehlermodell, nämlich dem ***Haftfehlermodell*** alle Fälle hinreichend gut erfaßt werden können. Dieses Modell nimmt immer dann die Entdeckung eines Fehlers an, wenn ein Knoten sich nicht 'bewegt', d.h. daß er entweder auf einem „0"- oder auf einem „1"-Pegel fest hängt. Damit besteht „nur" noch die Aufgabe, alle Knoten von außen beobachtbar zu machen. Das ist keineswegs eine triviale Angelegenheit, denn das Anfügen zusätzlicher Pins zu Testzwecken führt praktisch nie zum Ziel weil meist eine enorme Anzahl nötig wäre, die die Gehäusekosten in untragbare Höhe treiben würde. Die heute meistens praktizierte Lösung ist dagegen der Einbau eines „langen", parallel ladbaren Schieberegisters, über das mit einem Pin (D-IN) sehr lange Testvektoren an die zu untersuchenden Knoten gebracht und über einen weiteren Pin (D-OUT) die Reaktionen seriell ausgelesen werden können. Diese Vorgehensweise ist als SCAN-Path-Technik bekannt. Andere Begriffe dafür sind ***Boundary Scan Technik*** oder JTAG[18]-Interface. Die JTAG–Standardisierung wird später noch im Zusammenhang mit der FPGA–Konfigurierung angesprochen. Sie ist nicht ohne Grund in viele Bereiche der Digitaltechnik eingeflossen. Denn durch die einfache Erreichbarkeit aller internen Knoten einer Schaltung über nur zwei Pins ergeben sich vielfältige Möglichkeiten, die über den Test hinaus auch zur Emulation genutzt werden können.

Die Komplexität heutiger integrierter Digitalschaltungen macht einen vollständigen Test, bei dem jeder Knoten einmal 'getoggelt' werden müßte, um festzustellen ob er fest hängt, unmöglich. Um die Problematik zu verdeutlichen, vergegenwärtige man sich, daß schon einfache Digitalschaltungen über mehrere hundert Flip-Flops verfügen. Diese können insgesamt 2^{Anzahl} Zustände speichern, die in einem vollständigen Test alle eingenommen werden müßten. Selbst wenn die Anzahl der Flip-Flops nur 200 wäre, müßten schon über $1{,}6 \cdot 10^{60}$ Zustände überprüft werden. Die Fehlersimulation hat deswegen die Aufgabe, mit Rechnerunterstützung und der Intuition des Ingenieurs den richtigen Kompromiß für einen handhabbaren, d.h., nicht zu umfangreichen Satz von Testvektoren zu finden. Die Rechnerhilfe besteht im wesentlichen darin, daß ein Programm an zufällig gewählten Stellen der Schaltung einzelne Haftfehler einbaut und dann prüft, ob jeder dieser Fehler mit dem Testvektorsatz gefunden werden kann. Falls nicht, ist der Ingenieur am Zug und muß auf der Basis der Kenntnis seines Entwurfs gezielt weitere Testvektoren hinzufügen und natürlich stets den Erfolg überprüfen. Es ist leicht vorstellbar, daß diese Prozedur auch bei hoher Rechenleistung sehr lange dauern kann, bis schließlich ein statistisch verläßlicher Fehlerentdeckungsgrad, der nahe an 90% liegen sollte, erreicht ist.

Obwohl Fehlersimulation und Testvektorerzeugung mit dem Entwurf direkt nichts zu tun haben, fallen diese Aufgaben dennoch dem Schaltungsentwickler zu, weil kein anderer sie besser lösen kann. In der Praxis kann das bedeuten, daß bei komplexen Schaltungen mehr als die Hälfte der Entwicklungszeit allein für den Test aufgewendet werden muß. Die Scheu vor diesem hohen Aufwand hat nicht zuletzt zu der wachsenden Beliebtheit von FPGAs geführt, für die der Halbleiterhersteller den Test ja bereits durchgeführt hat und somit für fehlerfreie Hardware garantiert.

An dieser Stelle sei aber auf eine Falle hingewiesen, in die man als Entwickler geraten kann, wenn man sich von vornherein alle Überlegungen hinsichtlich Test und Testbarkeit einer Schaltung erspart. Der Begriff 'Design for Testability' besagt, daß es wichtig ist, von Anfang an einen möglicherweise später benötigten Test im Auge zu behalten. Trifft man diese Vorsorge nicht, kann eine Schaltung entstehen, die gar nicht testbar ist. Wird dann überraschend der Test trotzdem benötigt, etwa weil mit wachsender Stückzahl ein Umschwenken vom FPGA auf Standardzellen geboten ist, dann können grundlegende Änderungen anfallen, die den Entwickler in eine frühe Phase des Designs zurückversetzen. Er kann dabei weit mehr Zeit verlieren, als wenn er von Beginn an – auch für das FPGA – die Testbarkeit bei jedem Entwurfsschritt sichergestellt hätte.

[18] **J**oint **T**est **A**ction **G**roup

6.2 VHDL–basierte Realisierung einfacher digitaler Schaltungen

6.2.1 Modellbeispiele

Einführungsbeispiel: NOR-Gatter

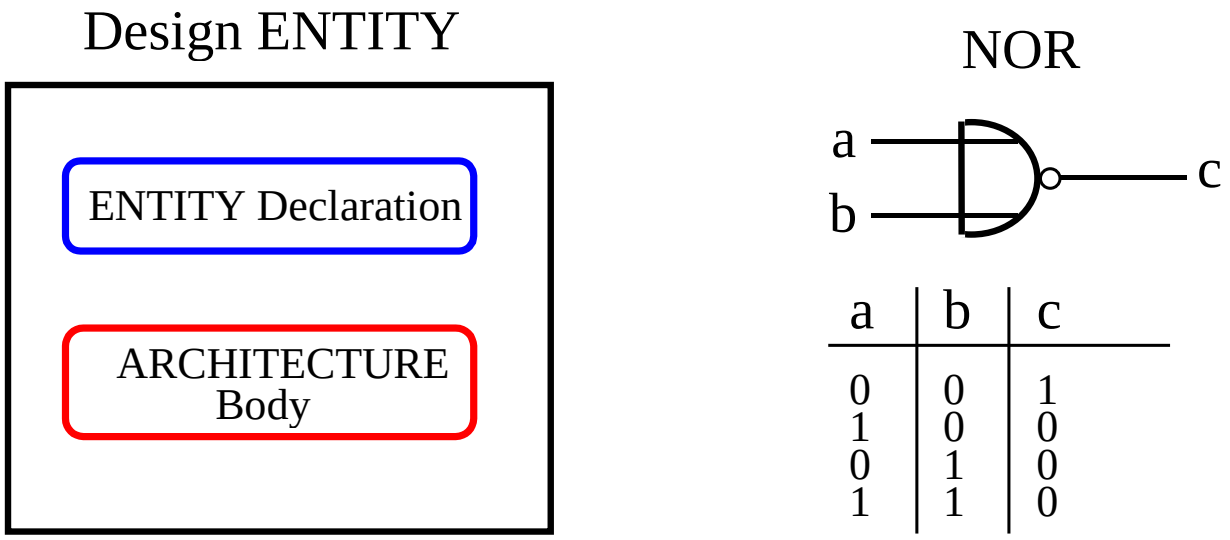


Bild 6.10: Entity und Architecture als Grundelemente eines VHDL–Modells

```
ENTITY NOR_Gate IS
PORT (a, b:  IN bit;
             c:      OUT bit)
END NOR_Gate;
```

```
ARCHITECTURE Verknuepfung OF NOR_Gate IS
BEGIN
c<= '1' WHEN a = '0' AND b = '0' ELSE
          '0' WHEN a = '0' AND b = '1' ELSE
          '0' WHEN a = '1' AND b = '0' ELSE
          '0' WHEN a = '1' AND b = '1' ELSE
          '0'
END Verknuepfung
```

```
BEGIN
c <= a NOR b
END Verknuepfung
```

← alternativ

4bit-Addierer

```
ENTITY Adder IS

PORT (a, b:   IN INTEGER RANGE 0 TO 15;
      c:      OUT INTEGER RANGE 0 TO 15)

END Adder;
```

```
ARCHITECTURE Addition OF Adder IS
BEGIN
c <= a + b;
END Addition;
```

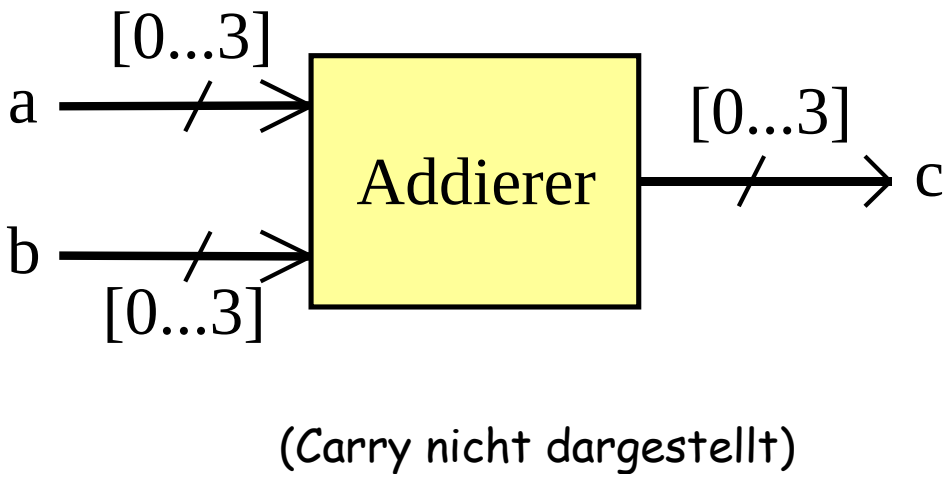


Bild 6.11: Die Entity des Addierers als Blockschaltbild

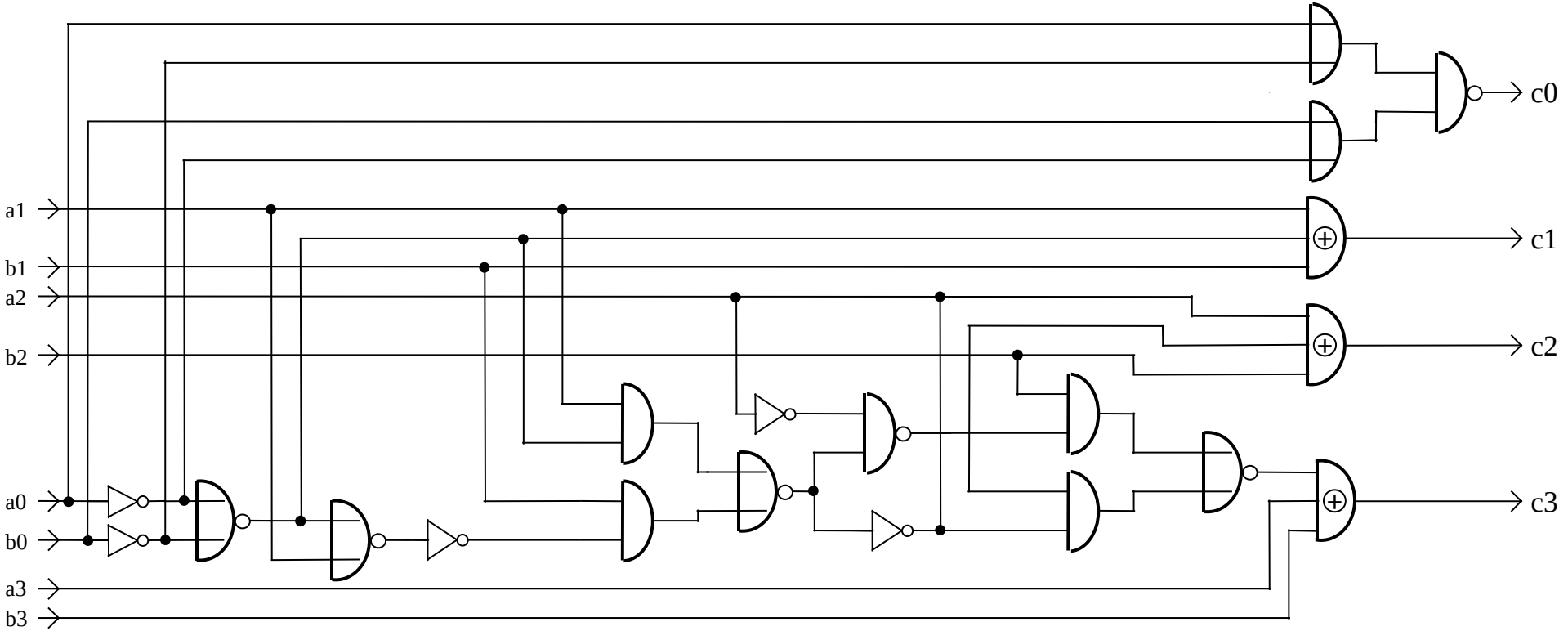


Bild 6.12: Ergebnis der VHDL–Synthese für den 4bit-Addierer

- Probleme von Beschreibungen auf algorithmischer Ebene

Der folgende Ausschnitt aus der Architektur eines hier nicht näher spezifizierten Schnittstellenbausteins ist eine Beschreibung auf algorithmischer Ebene. Der Baustein soll immer dann, wenn er von einem Controller eine Aufforderung `adr_request` erhält, eine interne Adresse `int_adr` frühestens nach 10 ns auf den Bus, d.h. auf `bus_adr` legen. Die Beschreibung enthält keine Angaben über die Schaltungsstruktur und weder Takt- noch Rücksetzsignale.

```
ARCHITECTURE algo_ebene OF io_ctrl IS
BEGIN
  ....
  schreib_dat_alg: PROCESS
  BEGIN
            WAIT UNTIL adr_request = '1';
            WAIT FOR 10ns;
            bus_adr <= int_adr;
  END PROCESS schreib_dat_alg;
END algo_ebene;
```

Register-Transfer-Ebene

```
ARCHITECTURE rt_ebene OF io_ctrl IS
BEGIN
          .....
           SR_DAT: PROCESS (clk)
                VARIABLE zwischen: boolean;
           BEGIN
                IF rising_edge(clk) THEN
                   IF ((adr_request = '1') AND (zwischen = false)) THEN
                   zwischen:= true;
                   ELSIF (zwischen = true) THEN
                         bus_adr <= int_adr;
                         zwischen := false;
                   END IF;
                END IF;
           END PROCESS SR_DAT;
END rt_ebene;
```

Um das Beispiel auf der Register-Transfer-Ebene darzustellen, wurde ein Taktsignal `clk` hinzugefügt, so daß die Abläufe in Abhängigkeit von diesem Signal beschrieben werden können.

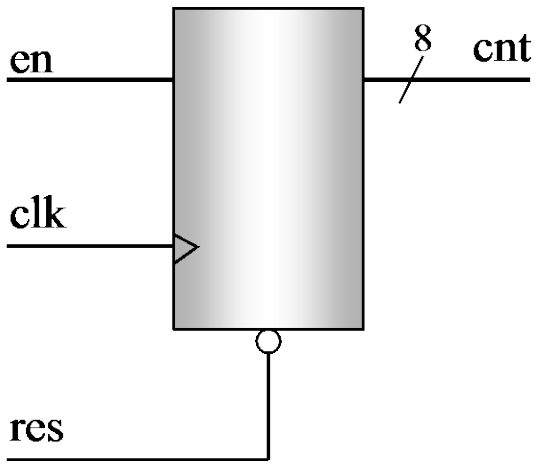
Wenn bei einer **ersten** aktiven Taktflanke `clk` ein gesetztes `adr_request`-Signal entdeckt wird, wird zunächst die temporäre Variable `zwischen` gesetzt, damit bei der **nächsten** aktiven Taktflanke – **Wartezeit** – die Adresse auf den Bus geschrieben werden kann. Durch geeignete Wahl der **Taktperiode** ist sicherzustellen, daß die Wartezeit von mindestens 10 ns eingehalten wird. Im Gegensatz zur algorithmischen Ebene wird hier ein zeitliches Schema für den Ablauf vorgegeben und implizit eine Schaltungsstruktur beschrieben.

- 8bit-Vorwärts-Binärzähler mit asynchronem Reset und Enable

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
```

```
entity Zaehl is
port (clk, res, en : in std_logic;
cnt: out std_logic_vector(7 downto 0);
end Zaehl;
```

```
architecture COUNT of Zaehl is
signal icnt: std_logic_vector(7 downto 0);
begin
process (clk, en, icnt, res)
        begin
        if (res = '0') then
        icnt <= (others => '0');
        elsif (clk'event and clk='1') then
    if (en = '1') then
        icnt <= icnt+'1';
end if;
     end if;
end process;
        cnt <= icnt
end COUNT
```



Finite State Machine

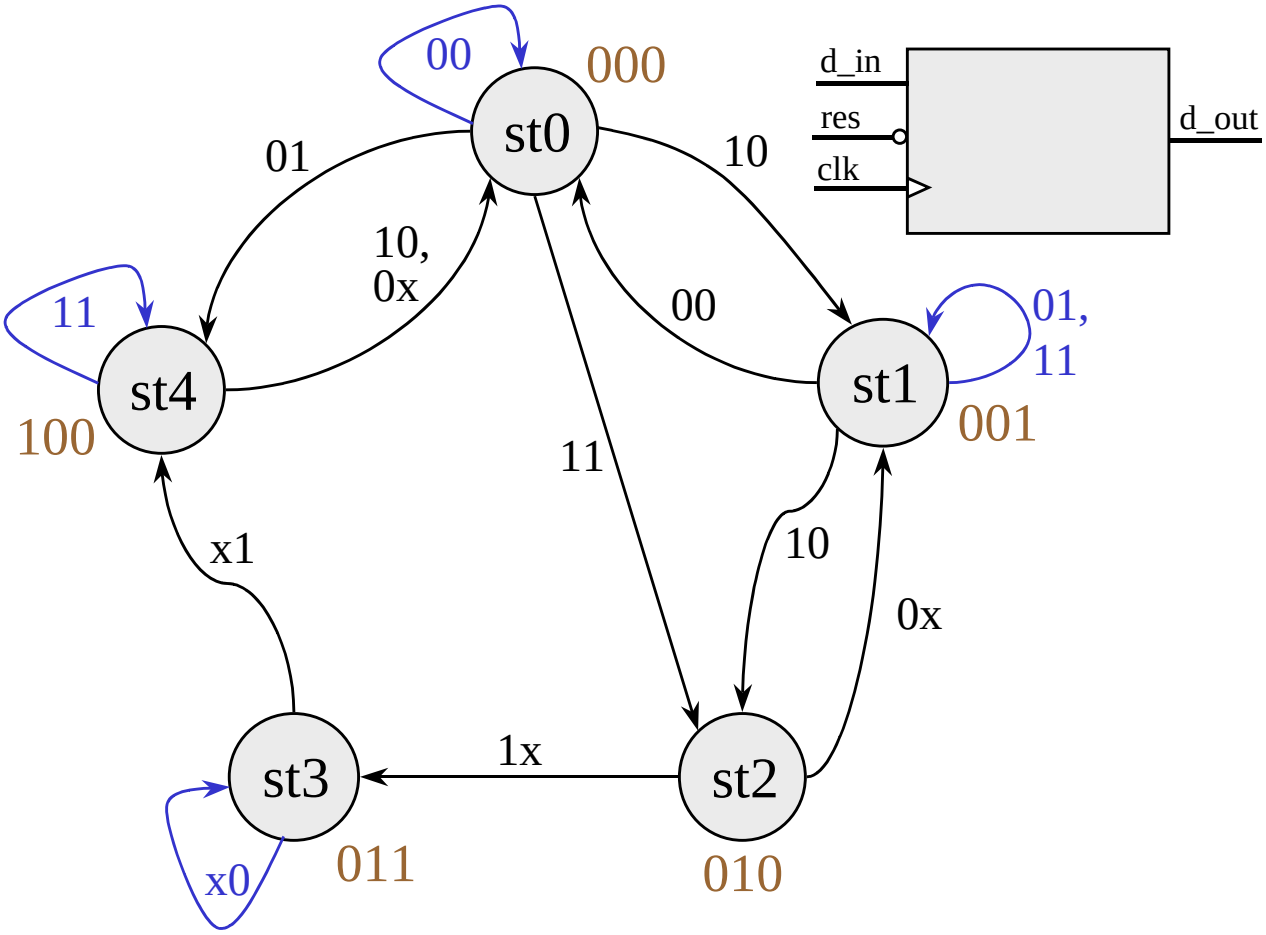


Bild 6.13: FSM-Beschreibung eines MOORE-Automaten

VHDL-Beschreibung

```
library IEEE;
use ieee.std_logic_1164.all;
```

```
entity M_AUT is
port (clk, res: in std_logic;
d_in: in std_logic_vector (1 downto 0);
d_out: out std_logic_vector (2 downto 0);
end M_AUT;
```

```
architecture FUNCT of M_AUT is
        type zustands_werte is (st0, st1, st2, st3, st4);
        signal akt_zust, folg_zust: zustands_werte;
        begin
   -- Definition der FSM-Register
Zust_Reg: process (clk, res)
        begin
        if (reset = '0') then
        akt_zust <= st0;
```

```
            elsif (clk ='1' and clock'event) then
            akt_zust <= folg_zust;
        end if;
end process Zust_Reg;
```

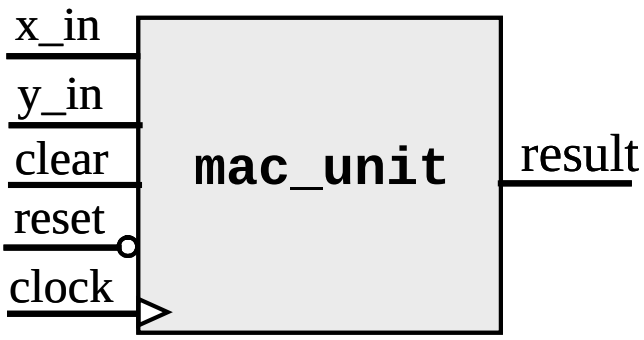
```
-- Kombinatorik der FSM

FSM: process (akt_zust, d_in)
begin
case akt_zust is

        when st0 =>
        case d_in is
            when    “00”    => folg_zust <= st0;
            when    “01”    => folg_zust <= st4;
            when    “10”    => folg_zust <= st1;
            when    “11”    => folg_zust <= st2;
            when    others => null;
end case;

        when st1 =>
        case d_in is
            when    “00”        => folg_zust <= st0;
            when    “10”        => folg_zust <= st2;
            when    others  => folg_zust <= st1;
end case;
        when st2 =>
        case d_in is
            when    “00”        => folg_zust <= st1;
            when    “01”        => folg_zust <= st1;
            when    “10”        => folg_zust <= st3;
            when    “11”        => folg_zust <= st3;
            when    others      => null;
end case;
        when st3 =>
        case d_in is
            when    “01”        => folg_zust <= st4;
            when    “11”        => folg_zust <= st4;
            when    others      => folg_zust <= st3;
end case;

        when st4 =>
        case d_in is
            when    “11”          => folg_zust <= st4;
            when    others        => folg_zust <= st0;
end case;

        when others => folg_zust <= st0;
end case;
end process FSM;

    -- Moore-Ausgabewerte (hängen nur von akt_zust ab)

AUSGABE: process (akt_zust)
```

```
        begin
        case akt_zust is
        when st0 => d_out <= "000";
        when st1 => d_out <= "001";
        when st2 => d_out <= "010";
        when st3 => d_out <= "011";
        when st4 => d_out <= "100";
        when others => d_out <= "000";
end case;
end process AUSGABE;
end FUNCT;
```

VHDL-Beschreibung einer MAC-Unit

```
ENTITY mac_unit IS
PORT (clock, reset, clear : IN bit;
x_in, y_in          : IN integer range -8 to 7; -- 4bits
result              : OUT integer range -128 to 127); -- 8bits
END mac_unit;
```

```
ARCHITECTURE mac_calc OF mac_unit IS
SIGNAL     int_bus: integer range -128 to 127;
BEGIN
     PROCESS (clock, reset)
     BEGIN
     IF reset  = '0' THEN
     int_bus <= '0';
     ELSEIF clock= '1' AND clock'event  THEN
     IF clear  = '1' THEN
     int_bus  <= int_bus + x_in * y_in;
     ELSE
     int_bus <= x_in * y_in;
     END IF;
     END IF;
     END PROCESS;
result <= int_bus;
END  mac_calc;
```



6.3 Einführung in den Gebrauch von VHDL

6.3.1 Allgemeines

Für die allgemeine Akzeptanz einer HDL ist ihre Effizienz im Rahmen einer Entwicklungsumgebung von entscheidender Bedeutung.

Die HDL muß werkzeugunabhängig sein.

VHDL erfüllt diese Voraussetzungen und hat darüber hinaus den Status eines IEEE-Standards. Auf dieser soliden Grundlage hat VHDL einen hohen Grad der Durchdringung in Industrie und Forschung erreicht. Eine Vielzahl von Werkzeugen für Simulation und Synthese stehen heute auf der Basis von VHDL zur Verfügung. Im folgenden sind einige wichtige Aspekte, deren sich jeder Nutzer von VHDL bewußt sein sollte, zusammengestellt.

- ✓ VHDL ist sowohl von der Hardwareplattform als auch von der dort verwendeten Software (z.B. Betriebssystem) unabhängig. Es gibt mittlerweile zahlreiche Anbieter von Software, die für die meisten Entwurfsschritte digitaler integrierter Schaltungen VHDL-basierte Lösungen bereitstellen. Da VHDL das Einbinden systemabhängiger Aspekte in 'Packages' unterstützt, können VHDL-Modelle in der Regel problemlos portiert werden.
- ✓ VHDL ist technologieunabhängig. Die Entscheidung für eine bestimmte Technologie (FPGA, Gate Array, Standardzelle, Vollkundenschaltung) muß erst relativ spät getroffen werden. Ein eventuell nötiges Umschwenken auf eine andere Technologie erfordert nur Änderungen in den technologieorientierten Schritten, d.h. beginnend mit der Schaltungssynthese.
- ✓ VHDL hat das Entstehen und die starke Verbreitung leistungsfähiger Synthesewerkzeuge in Gang gesetzt. Auf diese Weise ist eine neue und produktivere Entwurfsmethodik für digitale integrierte Schaltungen heute jedem Ingenieur an seinem Laborarbeitsplatz zugänglich.
- ✓ VHDL ist ohne Erläuterungen durch Kommentare les- und verstehbar. Die Syntax ist sehr ausführlich und hat selbsterklärende Befehle, so daß der Begriff Selbstdokumentation berechtigt ist.
- ✓ Mittels VHDL läßt sich ein bedeutendes Merkmal anwendungsspezifischer Digitalschaltungen, nämlich das parallele Arbeiten von Funktionseinheiten, beschreiben, wohingegen nahezu alle heutigen Standard-Prozessoren immer noch sequentiell orientiert sind.
- ✓ Als Nachteil von VHDL ist die Komplexität zu nennen, die eine lange Einarbeitungszeit erforderlich macht. Das Verhalten komplexer VHDL-Modelle in der Simulation ist für einen Neuling kaum nachvollziehbar.

VHDL-Sprachumfang

Die in Rahmen dieser Einführung beschriebenen Anweisungen sind eine Untermenge der Möglichkeiten, die VHDL bietet. So wird beispielsweise nicht auf Funktionen und Prozeduren eingegangen. Auch Befehle wie „**EXIT**“ oder „**ASSERT**“ sind für diese Einführung nicht von Bedeutung und wurden deshalb weggelassen.

6.3.2 Port-Deklaration

In der Port-Deklaration werden jeweils die Ein- und Ausgänge eines Entwurfes deklariert. Der Modus gibt dabei die Richtung des Signals an. IN bedeutet, daß es sich bei dem Signal um ein Eingangssignal handelt. Solche Signale sind daher nicht beschreibbar.

OUT dagegen zeigt an, daß ein Ausgangssignal vorliegt. Solche Signale sind daher nur beschreibbar und nicht lesbar. Sie können auch nicht in logische Verknüpfungen einbezogen werden. Durch INOUT werden Busse deklariert.

’Typ’ gibt den Typ des Signals an. Dieser kann vordefiniert oder ’selbstdefiniert’ sein. Bei der Syntax ist zu beachten, daß die letzte Zeile nicht mit einem Semikolon abgeschlossen wird.

Syntax:

```
PORT (port_name: Modus Typ;
      port_name: Modus Typ
     );
```

Beispiel:

```
PORT (a: IN    BOOLEAN;
      b: IN    INTEGER RANGE 0 TO 31;
      c: INOUT BIT;
      e: OUT   STD_LOGIC;
      f: OUT   STD_LOGIC_VECTOR (3 DOWNTO 0)
     );
```

6.3.3 Konstantendeklaration

Eine Konstante weist einem Namen einen bestimmten Wert zu. Dieser Wert kann im gesamten Programm nur gelesen und nicht neu beschrieben werden.

Es ist vorteilhaft, ein Programm soweit wie möglich in Abhängigkeit von Konstanten zu entwickeln (z.B. Länge eines Schieberegisters, Breite eines Addierers). Dies erleichtert spätere Änderungen erheblich.

Syntax:

```
CONSTANT Konstantenname: Typ := Wert;
```

Beispiel:

```
CONSTANT a: BOOLEAN   := TRUE;
CONSTANT b: INTEGER   := 31;
CONSTANT c: BIT_VECTOR (3 DOWNTO 0) := "0000";
CONSTANT d: STD_LOGIC := 'Z';
CONSTANT e: STD_LOGIC_VECTOR (3 DOWNTO 0) := "0-0-";
```

6.3.4 Variablendeklaration

Variable haben im Gegensatz zu Signalen keine globale Gültigkeit. Sie können daher nur in Prozessen deklariert und verwendet werden. Es gibt zwar die Möglichkeit, globale Variablen in Form von **’SHARED VARIABLES’** zu definieren. Davon sollte jedoch möglichst kein Gebrauch gemacht werden, da Konstrukte entstehen können, deren Verhalten nicht vorhersagbar ist.

Auch hier können Defaultwerte zugewiesen werden. Ansonsten werden wie bei den Signalen die niedrigsten Werte des erlaubten Wertebereichs angenommen.

Syntax:

```
VARIABLE var_name: Typ;
```

Beispiel:

```
VARIABLE a: BOOLEAN;
VARIABLE b: INTEGER RANGE 0 TO 31;
VARIABLE c: BIT;
VARIABLE d: BIT_VECTOR (3 DOWNTO 0);
VARIABLE e: STD_LOGIC;
VARIABLE f: STD_LOGIC_VECTOR (0 TO 3);
```

6.3.5 SIGNALE (Einführung)

Neben Variablen und Konstanten, die auch von prozeduralen Programmiersprachen her bekannt sind, verfügt VHDL zusätzlich über Objekte der Klasse „Signal“. Diese Klasse wurde eingeführt, um spezielle Eigenschaften elektronischer Schaltungen modellieren zu können. Unter anderem können Änderungen von Signalwerten zeitlich verzögert zugewiesen werden. Damit lassen sich die Laufzeiten von Hardwarekomponenten nachbilden. Signale dienen im wesentlichen dazu, Daten zwischen parallel arbeitenden Modulen auszutauschen. Verschiedene Module können dabei auf ein und dasselbe Signal schreiben. Diese Eigenschaft gestattet die Modellierung von Bussen.

Wird bei der Deklaration kein Defaultwert mit **:=** explizit zugewiesen, so wird der bei Typ am weitesten links stehende Wert als Defaultwert angenommen.

Syntax:

```
SIGNAL sig_name: Typ;
```

Beispiel:

```
SIGNAL a: BOOLEAN := true;          --Defaultwert: true
SIGNAL b: BOOLEAN;                  --Defaultwert: false
SIGNAL c: INTEGER RANGE 0 TO 31;    --Defaultwert: 0
SIGNAL d: BIT;                      --Defaultwert: ‘0‘
          SIGNAL e: BIT_VECTOR (3 DOWNTO 0);  --Defaultwert:
                                                ‘0‘,‘0‘,‘0‘,‘0‘
SIGNAL f: STD_LOGIC;                --Defaultwert: ‘0‘
```

6.3.6 Zuweisungen

Signalzuweisung:

```
sig_name <= Wert;
```

Variablenzuweisung:

```
var_name := Wert;
```

Bemerkung: Durch die Symbole **:=** oder **=:** bzw. **=>** oder **<=** werden Wertzuweisungen an Variablen und an Signale unterschieden.

6.3.7 Komponenten

Komponenten dienen der Realisierung strukturaler Modelle. Durch Einbinden getrennter Programmstücke als 'BLACK BOXES' wird das Gesamtprogramm sehr übersichtlich. Man muß nur spezifizieren, wie Ein- und Ausgangssignale verdrahtet werden, ohne den Inhalt der entsprechenden Komponente anzuschauen. Ein Ändern von Funktionen ist dadurch selbst in späten Entwicklungsstadien einfach möglich.

6.3.7.1 Komponentendeklaration

Eine Deklaration macht eine Komponente mit ihren Ein- und Ausgangsports bekannt. Dies geschieht im Deklarationsteil einer **ARCHITECTURE**, eines **BLOCKS** oder einer **PACKAGE**.

Syntax:

```
COMPONENT Komponentenname
  Portdeklarationen;
END COMPONENT;
```

6.3.7.2 Komponenteninstantiierung

Bei der Instantiierung werden die bereits bekannt gemachten Ports der Komponente verdrahtet. Dies geschieht durch Zuweisen von Signalnamen an die Ports.

Syntax:

```
Name: Komponentenname
   PORT MAP (
     port_name => sig_name,
     port_name => sig_name
            );
```

6.3.8 Typdefinition

Bevor in einem VHDL-Modell mit einem Objekt gearbeitet werden kann, muß festgelegt werden, welche Werte das Objekt annehmen darf. Zu diesem Zweck werden Datentypen deklariert, mit deren Hilfe die Wertebereiche definiert werden. VHDL verfügt nur über sehr wenige, vordefinierte Datentypen, wie z.B. **`real`** oder **`integer`**. Es gibt jedoch umfangreiche Möglichkeiten, benutzerdefinierte Datentypen zu erzeugen.

Man kann von vordeklarierten Typen Untertypen ableiten. Untertypen sind im einfachsten Fall im Wertebereich eingeschränkte Grundtypen. Die weitere Ableitung von Untertypen aus einem Untertyp ist nicht möglich.

Beispiel:

```
TYPE Aufzaehlungstyp IS (a, b, c, d, e);
SUBTYPE S_Aufzaehlungstyp IS Aufzaehlungstyp RANGE b TO d;
```

6.3.9 PROCESS (Einführung)

Während nebenläufige Anweisungen direkt im Programmteil der **Architecture** stehen, enthalten die ebenfalls dort angesiedelten Prozesse Vorgänge, die in Abhängigkeit von Signalereignissen ablaufen, wie z.B. Taktvorderflanken.

Hier ist zu beachten, daß ein Prozeß nur dann aufgerufen wird, wenn sich eines der in seiner 'Sensitivity-Liste' aufgeführten Signale ändert.

Prozesse sind generell nebenläufig.

Während Komponenten in Prozessen nicht erlaubt sind, dürfen aber lokale Variablen dort deklariert und unmittelbar verwendet werden.

Syntax:

```
Name:                              -- optional
  PROCESS (Sensitivity-List)
    Deklarationen
  BEGIN
    Anweisungen
  END PROCESS;
```

Beispiel:

```
PROCESS (clock)
BEGIN
  IF (clock'event AND clock='1') THEN -- Taktvorderflanke
    ...
  END IF;
END PROCESS;
```

6.3.10 IF-Anweisung

Die IF-Anweisung ist ebenfalls aus höheren Programmiersprachen bekannt. Sie arbeitet Anweisungen in Abhängigkeit von einer Bedingung ab, sobald diese als Ergebnis den booleschen Wert **true** liefert.

Syntax:

```
IF Bedingung THEN Anweisung;
   ELSIF Bedingung THEN Anweisung
   ELSE Anweisung
END IF;
```

6.3.11 CASE-Anweisung

Die CASE-Anweisung ist eine Möglichkeit zur Abkürzung von IF-Befehlsketten bei bedingter Ausführung mehrerer Befehle, die alle abhängig von einem Ausdruck sind. 'WHEN OTHERS' ist ein Synonym für alle nicht aufgeführten Bedingungen.

Syntax:

```
CASE Ausdruck IS
  WHEN wert   => Anweisung;
  WHEN wert   => Anweisung;
  WHEN OTHERS => Anweisung;
END CASE;
```

Beispiel:

```
CASE sel IS
  WHEN 0 | 1 | 2 => z <= b;
  WHEN 3 to 10   => z <= c;
  WHEN OTHERS    => z <= d;
END CASE;
```

6.3.12 Bedingte Signalzuweisung

Während die CASE-Anweisung in Abhängigkeit von einem Ausdruck entscheidet, können mit der bedingten Signalzuweisung einem Signal in übersichtlicher Weise mehrere Werte in Abhängigkeit von mehreren Bedingungen zugewiesen werden.

Fall 1

Syntax:

```
Name:     -- optional
sig_name <= Wert WHEN Bedingung ELSE
            Wert WHEN Bedingung ELSE
            Wert;
```

Beispiel:

```
z <= a WHEN (x>10) ELSE
     b WHEN (x>5)  ELSE
     c;
```

Fall 2

Syntax:

```
Name:     -- optional
  WITH Ausdruck   SELECT
    sig_name <=   Wert WHEN Bedingung,
                  Wert WHEN Bedingung,
                  Wert WHEN OTHERS;
```

Beispiel:

```
WITH sel SELECT
  z <= a WHEN 0 | 1 | 2,
       b WHEN 3 TO 10,
       c WHEN OTHERS;
```

6.3.13 Die FOR-Schleife

Die FOR-Schleife wird {Obergrenze – Untergrenze + 1}-mal sequentiell abgearbeitet, d.h. der Ablauf entspricht dem äquivalenten Konstrukt einer höheren Programmiersprache. Eine lokale Variable *i* muß dabei nicht vorher deklariert worden sein. Dies geschieht automatisch beim ersten Aufruf der Schleife. Die Anzahl der Schleifendurchläufe hingegen muß für die Synthese bekannt sein.

Syntax:

```
Name:  -- optional
  FOR i IN Untergrenze TO Obergrenze LOOP
    Anweisungen
  END LOOP;
```

Beispiel:

```
FOR i IN 1 TO 10 LOOP
  a(i):=i*i;
END LOOP
```

6.3.14 Die WHILE-Schleife

Auch die WHILE-Schleife ist aus herkömmlichen Programmiersprachen bekannt. Die Schleifenbedingung wird dabei am Beginn der Schleife geprüft. Die WHILE-Schleife kann daher gegebenenfalls ohne Ausführung übersprungen werden.

Syntax:

```
WHILE Bedingung LOOP
      Anweisung
END LOOP;
```

Beispiel:

```
i:=0;
WHILE (i<10) LOOP
  s<=i;
  i:=i+1;
END LOOP;
```

6.3.15 WAIT-Anweisung

Die WAIT-Anweisung bewirkt eine Pause bis zum Eintreten eines Ereignisses. Dies kann eine Bedingung sein, eine Signaländerung oder einfach der Ablauf einer bestimmten vorgegebenen Pausendauer. WAIT kann an jeder Stelle eines Programms auftreten. Es ist jedoch zu beachten, daß dieser Befehl **nicht synthetisiert** werden kann. WAIT wird daher vorwiegend beim Erstellen von Testbenches benutzt, wodurch ein sequentieller Ablauf mit klar definierter Geschwindigkeit erzielt wird. Das ist eine Grundvoraussetzung für jegliche Simulation.

Beispiele:

```
WAIT ON a,b;          --(wartet bis a oder b sich ändert)
WAIT UNTIL x>10;      --(wartet bis x>10 ist)
WAIT FOR 10 ns;       --(stoppt das Programm für 10 ns)
WAIT;                 --(Endlosschleife)
```

6.3.16 Die GENERATE-Anweisung

Wird ein Hauptmodul aus mehreren gleichen Untermodulen aufgebaut, so lassen sich seine Strukturen mit Hilfe der GENERATE-Anweisung sehr übersichtlich gestalten. Abhängig von einer Bedingung oder einem über eine Schleife definierten Bereich wird eine Reihe von Anweisungen ein- oder mehrfach ausgeführt. Das untenstehende Beispiel verdeutlicht diesen Sachverhalt. Es wird ein Addierer aus mehreren Volladdierern und einem Halbaddierer aufgebaut. Dabei werden die Überträge übernommen und die Überläufe an den nächsten Block weitergegeben.

Syntax:

```
Name:
  FOR i IN Untergrenze TO Obergrenze GENERATE
    Anweisungen
  END GENERATE;

  IF Bedingung GENERATE
    Anweisungen
  END GENERATE;
```

Beispiel:

```
g0: FOR i IN 0 TO 3 GENERATE
      g1: IF i=0 GENERATE
            ha: half_adder PORT MAP (
                a => x(i), b => y(i),
                s => z(i), co => tmp(i+1)
                  );
          END GENERATE;
      g2: IF i>=1 AND i<=3 GENERATE
            fa: full_adder PORT MAP (
                a => x(i), b => y(i), c => tmp(i),
                s => z(i), co => tmp(i+1)
                  );
          END GENERATE;
    END GENERATE;
```

6.3.17 Die BLOCK-Anweisung

Eine BLOCK-Anweisung enthält einen Satz nebenläufiger Programmteile und ist bei der Organisation eines Entwurfes nützlich. Blocks können geschachtelt werden, um eine Hierarchie zu erzeugen. Objekte, die in einem Block deklariert sind, sind in diesem und in allen untergeordneten Blöcken bekannt.

Syntax:

```
Blockname: BLOCK
BEGIN
  Anweisungen
END BLOCK;
```

6.4 Aufbau eines VHDL-Modells

Ein VHDL-Modell besteht mindestens aus einer Schnittstellenbeschreibung (**Entity**), einer oder mehreren Verhaltens- oder Strukturbeschreibungen (**Architecture**) und gegebenenfalls einer oder mehreren Konfigurationen (**Configuration**). Die Aufteilung der Einheiten auf eine oder mehrere Dateien ist willkürlich. Beim Compilieren muß stets die Reihenfolge

Entity => Architecture => Configuration

beachtet werden.

Werden bestimmte Objekttypen, Funktionen oder Prozeduren von mehreren Modellen benötigt, ist es vorteilhaft, sie in einer übergeordneten Einheit (**Package**) abzulegen.

6.4.1 Bibliotheken (LIBRARIES)

Erfolgreich compilierte VHDL-Modelle werden in einer **Library** für die nachfolgende Simulation oder für die Weiterverwendung in anderen Modellen gespeichert.

Der Vorteil bei der Nutzung von Bibliotheken besteht darin, daß bereits verfügbare Schaltungsentwürfe nicht explizit von allen Anwendungsprogrammen eingebunden werden müssen. Ohne besondere Deklaration steht immer die **Library** **`*.std`** zur Verfügung. Hier sind die allgemeinen Packages gespeichert. Nach korrektem Compilieren eines VHDL-Quellcodes wird das Ergebnis einer eigenen Bibliothek mit dem Default-Namen **`work`** hinzugefügt.

Syntax für eine Bibliothekdeklaration:

- **`LIBRARY library_name_1 {,library_name_2...};`**

6.4.1.1 USE-Anweisung

Mit Hilfe der **Use**-Anweisung werden in Bibliotheken vorhandene Elemente angesprochen und stehen zur Nutzung für den aktuellen Entwurf zur Verfügung. Die jeweilige **Library** und gegebenenfalls die **Package**, in der die gewünschten Funktionen und Modelle abgelegt sind, muß angegeben werden. Auch ohne **Use**-Anweisung sind alle Elemente von **`*.std`** immer „sichtbar“.

Syntax:

```
USE library_name.all;
USE library_name.element_name;
USE library_name.package_name.all;
USE library_name.package_name.element_name;
```

6.4.2 PACKAGE

Eine **Package** enthält Anweisungen wie Konstanten- und Typendeklarationen sowie die Beschreibung von Prozeduren und Funktionen, die in mehreren VHDL-Modellen gebraucht werden. Zum Beispiel kann in einer **Package** der zu verwendende Logiktyp (zwei- oder mehrwertige Logik) mit allen korrespondierenden Operatoren definiert werden.

So können häufig benötigte Deklarationen durch einmalige Eingabe in verschiedene Projekte eingebunden werden. Packages eignen sich auch, um globale Informationen innerhalb eines komplexen Entwurfs oder innerhalb eines Projekts einmalig festzulegen und damit die Gefahr von widersprüchlichen Definitionen praktisch auszuschließen. Ein Ändern solcher globalen Informationen führt mit minimalem Aufwand zu einer Neukonfiguration eines kompletten Modells. Durch einfaches Ändern

der Wortbreite z.B. von Bussen, Multiplexern, Addieren oder Multiplizierern läßt sich ein Entwurf in erheblichem Umfang verändern und somit leicht den unterschiedlichsten Anforderungen anpassen.

Syntax:

```
PACKAGE definitionen IS
  CONSTANT zeit: TIME := 1 ns;
  CONSTANT a: INTEGER:= 0;
  SUBTYPE int_subtype IS INTEGER RANGE -16 TO 15;
  TYPE int IS ARRAY (3 DOWNTO 0) OF int_subtype;
END definitionen;
```

6.4.3 ENTITY

Eine **ENTITY** definiert einen neuen Komponentennamen, Ein- und Ausgänge und die damit verbundenen Deklarationen sowie Unterprogramme und sonstige Vereinbarungen, die für alle dieser **ENTITY** zugeordneten Architekturen gelten sollen. Die Teilmodelle eines komplexen Entwurfs „kommunizieren“ sozusagen über die **ENTITY**, d.h. über die dort vorgegebene Schnittstellenbeschreibung. Die deklarierten Ein- und Ausgänge sind die **Ports** eines Modells. Name, Datentyp und Signalflußrichtung werden in der **ENTITY** definiert. Außerdem werden hier Parameter deklariert, die dem Modell übergeben werden können, die so genannten **GENERICS**. So kann beispielsweise die Breite von Ports festgelegt werden.

Mit jeder Port-Deklaration der **ENTITY** werden **Signale** gleichen Typs und gleichen Namens deklariert, die unter Beachtung des jeweiligen **Modus** des Ports in der **ENTITY** und den zugehörigen **Architekturen** verwendet werden können:

- Modus **IN**: die Portleitungen werden als Eingänge deklariert (nur lesbar)
- Modus **OUT**: die Portleitungen werden als Ausgänge deklariert (nur schreibbar)
- Modus **INOUT**: die Portleitungen können als Ein- und als Ausgänge verwendet werden (les- und schreibbar)
- Modus **BUFFER**: die Portleitungen sind Ausgänge, die lesbar, aber nur von einer Quelle beschreibbar sind

Zusätzliche Signale müssen in den jeweiligen **ARCHITECTURES** deklariert werden.

Syntax:

```
ENTITY signale IS
PORT (clock, reset: IN  STD_LOGIC;
      x:   IN   INTEGER;
      y:   IN   STD_LOGIC_VECTOR (7 DOWNTO 0);
      z:   OUT   INTEGER RANGE -8 TO 7
        -- kein Semikolon in der letzten Zeile,
        -- sondern erst nach der folgenden Klammer
     );
END signale;
```

6.4.4 ARCHITECTURE

Die **ARCHITECTURE** beschreibt das Verhalten einer Komponente. Dazu gibt es verschiedene Möglichkeiten, nämlich eine Struktur- oder eine Verhaltensbeschreibung. Beides ist auch kombinierbar.

Im Unterschied zu herkömmlichen Programmiersprachen dient VHDL der Hardwarebeschreibung. Hardwarestrukturen arbeiten sowohl nebenläufig als auch sequentiell. Um solche Eigenschaften zu erfassen, müssen nebenläufige Anweisungen zur Verfügung stehen, die parallel bearbeitet werden.

VHDL-Anweisungen können in parallele und sequentielle eingeteilt werden. Innerhalb des Anweisungsteils einer **ARCHITECTURE** sind Konstrukte wie z.B. die Instantiierung von Komponenten, **BLOCK**- und **GENERATE**-Anweisungen sowie sämtliche Prozesse nebenläufig. Sequentielle Anweisungen (z.B. zum Abarbeiten von Schleifen) hingegen funktionieren wie es von konventionellen Programmiersprachen bekannt ist. Sie dürfen nur innerhalb von Prozessen, Funktionen oder Prozeduren stehen.

Auf Basis einer **ENTITY** können mehrere Architekturen miteinander kombiniert werden, d.h. es können für eine Schnittstellendefinition mehrere Beschreibungen auf unterschiedlichen Abstraktionsebenen oder verschiedene Entwurfsalternativen existieren.

Die in ARCHITECTURES deklarierten Signale stellen die Verbindungen zwischen den einzelnen Strukturen her. Sie sind **nur** in der jeweiligen **ARCHITECTURE** bekannt. Von außen kann nicht darauf zugegriffen werden.

Die innerhalb von ARCHITECTURES spezifizierten Prozesse, auf die im folgenden eingegangen wird, sind grundsätzlich nebenläufig.

Syntax:

```
ARCHITECTURE signale_arch OF signale IS
SIGNAL a:    INTEGER RANGE -4 TO 3;
SIGNAL b, c: INTEGER;
BEGIN
B <= a;    -- nebenläufige Anweisungen (nicht getaktet)
C <= a+1;
P1: PROCESS (clock, reset)
  VARIABLE d: INTEGER;
  BEGIN
  ...
END PROCESS
P2: PROCESS (clock, reset)
  VARIABLE e: INTEGER;
  BEGIN
  ...
  END PROCESS
END signale_arch
```

6.4.5 PROZESSE (Vertiefung)

Während nebenläufige Anweisungen direkt im Programmteil der ARCHITECTURE stehen, enthalten die ebenfalls dort angesiedelten Prozesse Vorgänge, die in Abhängigkeit von Signaländerungen (d.h. ereignisgesteuert) gestartet werden (z.B. Taktflanke).

Ein Prozeß beschreibt das Modellverhalten algorithmisch. Nach dem Schlüsselwort **PROCESS** steht in Klammern die so genannte Empfindlichkeitsliste (Sensitivity List). Sie beinhaltet diejenigen Sig-

nale, auf deren Änderung der Prozeß wartet, d.h. er wird dann - und nur dann - aktiviert, wenn eine solche Änderung aufgetreten ist.

Prozesse sind nebenläufig, das heißt, sie können gleichzeitig ablaufen.

Während Komponentendeklarationen und Instantiierungen in Prozessen **nicht** erlaubt sind, dürfen hier lokale Variablen deklariert und unmittelbar anschließend verwendet werden.

Wird ein Taktsignal in die Empfindlichkeitsliste geschrieben, dann wird bei der Synthese ein Flip-flop generiert, das mit diesem Signal getaktet ist. Durch eine derartige zeitdiskrete Ablaufsteuerung entsteht zwar ein gewisser Schaltungsaufwand, jedoch werden durch die nunmehr synchrone Arbeitsweise klar definierte Zustände erzeugt.

Im folgenden Beispiel ist ein getakteter Prozeß dargestellt.

Syntax:

```
PROCESS (signale)
-- Variablendeklarationen
BEGIN
-- Anweisungen
END PROCESS;
```

Beispiel:

```
PROCESS (clock, reset)
-- ggf. Variablendeklaration
BEGIN
  IF reset ='0' THEN
      -- Anweisungen (im Falle eines Resets)
  ELSIF (clock'event AND clock='1')
      -- nebenläufige/sequentielle Anweisungen (getaktet)
  END IF;
END PROCESS
```

6.4.6 VARIABLEN (Vertiefung)

VARIABLEN sind Objekte, deren aktueller Wert gelesen und ohne Zeitverzögerung neu zugewiesen werden kann. Der Variablenwert kann sich also im Laufe der Simulation ändern. Variable existieren nur lokal innerhalb von Prozessen.

In ARCHITECTURES können **keine lokalen Variablen** deklariert werden. Dies geschieht ausschließlich innerhalb von Prozessen. Auf diese Prozesse ist dann auch die **Gültigkeit** der lokalen Variablen **beschränkt**.

Das folgende Beispiel erzeugt den in Tabelle 6.1 dargestellten Zeitverlauf. Dabei sei das Signal ***x*** von außen vorgegeben.

Beispiel:

```
PROCESS (clock)
VARIABLE y1, y2, y3: INTEGER;
BEGIN
  IF (clock'event AND clock='1') THEN -- Taktvorderflanke
    y1:=x;
    y2:=x+1;
    y3:=y2;
  END IF;
END PROCESS;
```

clock							
x	1	2	3	4	5	6	7
y1	1	2	3	4	5	6	7
y2	2	3	4	5	6	7	8
y3	2	3	4	5	6	7	8

Tabelle 6.1: Zeitverlauf bei Verwendung von Variablen

6.4.7 SIGNALE (Vertiefung)

Neben Variablen und Konstanten, die auch von prozeduralen Programmiersprachen her bekannt sind, verfügt VHDL zusätzlich über Objekte der Klasse **SIGNAL**. Diese Klasse wurde eingeführt, um spezielle Eigenschaften elektronischer Schaltungen modellieren zu können. Änderungen von Signalwerten können z.B. zeitverzögert zugewiesen werden **(*a* <= '0' after 10 ns)**. Damit lassen sich die Laufzeiten von Hardwarekomponenten nachbilden. Signale dienen im wesentlichen dazu, Daten zwischen parallel arbeitenden Modulen auszutauschen. Verschiedene Module können dabei auf ein und dasselbe Signal schreiben. Diese Eigenschaft gestattet die Modellierung von Bussen.

Signale übernehmen Werte **verzögert** nach ihrer Zuweisung. Diese Verzögerung wird auch 'Delta-Delay genannt. Sie entsteht durch die ereignisgesteuerte Simulation, die ein VHDL-Modell so oft abarbeitet, bis alle Signale stabil, d.h. alle Änderungen abgeklungen sind. Der stabile Zustand wird dann dem nächsten Schritt übergeben. Verzögerungen kommen bei getakteten Prozessen dadurch zustande, daß die im Prozeß festgelegten Schritte nur einmal beim Auftreten eines gemäß „Sensitivity List" definierten Ereignisses durchlaufen werden.

Zu beachten ist, daß in **ARCHITECTURES** deklarierten Signale nur dort (lokal) bekannt sind. Externe Modellbestandteile können nicht darauf zugreifen.

Im folgenden Beispiel ergeben sich durch Verwendung von Signalen Unterschiede im Zeitverlauf im Vergleich mit Tabelle 6.1, wo Variablen eingesetzt waren.

Beispiel:

```
SIGNAL x, y1, y2, y3: INTEGER;
...
PROCESS (clock)
BEGIN
  IF (clock'event AND clock='1') THEN -- Taktvorderflanke
    y1 <= x;
    y2 <= x+1;
    y3 <= y2;
  END IF;
END PROCESS
```

clock							
x	1	2	3	4	5	6	7
y1	'U'	1	2	3	4	5	6
y2	'U'	2	3	4	5	6	7
y3	'U'	'U'	2	3	4	5	6

Tabelle 6.2: Verlauf bei Verwendung von Signalen ('**U**' steht für nicht initialisiert, d.h. unbekannt)

Man erkennt, daß eine Entscheidung, ob Variablen oder Signale verwendet werden sollen, im Einzelfall sorgfältig abzuwägen ist.

6.4.8 Nebenläufige Anweisungen

Nebenläufige Anweisungen verzweigen nicht weiter in Unterkomponenten. Am Beispiel eines Halbaddierers werden die Vorteile dieser Art der Modellierung aufgezeigt. Da VHDL direkt in Hardware umgesetzt wird, sind später alle Abläufe tatsächlich parallel.

Im folgenden beschreibt ein komplettes VHDL-Modell einen Halbaddierer, der zwei Bit ***a*** und ***b*** addiert.

Beispiel:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY halfadd IS
PORT(a, b  : IN BIT;
     erg, c: OUT BIT
    );
END halfadd;

ARCHITECTURE halfadd_arch OF halfadd IS
BEGIN
  c <= a AND b;
  erg <= a XOR b;
END halfadd_arch;
```

6.4.9 Sequentielle Anweisungen

In sequentiellen oder auch prozeduralen Beschreibungen werden Konstrukte wie Verzweigungen (**IF, ELSIF, ELSE**), Schleifen (**LOOP**) und Unterprogrammaufrufe wie **FUNCTION** oder **PROCEDURE** verwendet. Die einzelnen Anweisungen werden dabei nacheinander (sequentiell) abgearbeitet. Solche Beschreibungen ähneln den Quellcodes höherer Programmiersprachen wie C oder Pascal.

Das folgende Beispiel stellt einen Addierer dar, der zwei 8-bit-Vektoren bitweise addiert und ständig das aktuelle Ergebnis ausgibt. An diesem Beispiel werden einige Schwierigkeiten deutlich, die eine Umsetzung sequentieller Strukturen in Hardware mit sich bringt. So kommen z.B. während des Rechenprozesses undefinierte Ausgaben vor. Erst nachdem der Addierer die Rechenoperation beendet hat, ist das Ergebnis stabil. Die Einführung eines Prozesses ist hier notwendig, weil die Schleife (**LOOP**) eine lokale Variable ***i*** deklariert – ein Vorgang, der nur in Prozessen erlaubt ist.

Beispiel:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY addierer IS
PORT(a, b : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
     erg : OUT STD_LOGIC_VECTOR (8 DOWNTO 0)
    );
END addierer;

ARCHITECTURE addierer_arch OF addierer IS
SIGNAL c: STD_LOGIC:='0';
BEGIN
PROCESS (a, b)
 FOR i IN 0 TO 7 LOOP
 erg(i) <= a(i) XOR b(i) XOR c;
c <= (a(i) AND b(i) AND NOT(c)) OR (NOT(a(i)) AND b(i) AND c)
      OR (a(i) AND NOT(b(i)) AND c);
 END LOOP;
 erg(8) <= c;
END PROCESS;

END addierer_arch;
```

6.4.10 Verhaltensmodellierung

Bei der Verhaltensmodellierung wird das Verhalten einer Komponente durch die Reaktion der Ausgangssignale auf die Eingangssignale unter Verwendung verschiedener Operatoren, Signalzuweisungen oder Bedingungen beschrieben.

Um in einer ARCHITECTURE Verhaltensmodellierung durchzuführen, sind Prozesse zu verwenden. Parallele Arbeitsweise kann durch mehrere Prozesse erreicht werden.

6.4.11 Strukturale Modellierung

Diese Modellierung bietet den Vorteil, daß in der Regel kein umfangreiches zusammenhängendes Programm entsteht, sondern ein Entwurf auf mehrere kurze Einheiten aufgeteilt werden kann, die getrennt kompiliert werden. Somit bleibt die Übersicht erhalten, und kleine Änderungen erfordern keinen hohen Zeitaufwand. In Top-Down-Entwürfen verwendet man gerne die strukturale Modellierung, weil bereits kompilierte Designs damit leicht eingebunden werden können. Es entstehen gut überschaubare hierarchische Strukturen in Form von Bäumen.

Beispiel:

Der Einfachheit halber entsprechen im folgenden Beispiel die Modelle XOR2 und AND2 den Operatoren XOR und AND.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE WORK.ALL;

ENTITY  halfadd IS
PORT (a, b:   IN BIT;
      erg, c: OUT BIT
     );
END halfadd;

ARCHITECTURE halfadd_arch OF halfadd IS

COMPONENT XOR2
PORT(x, y: IN BIT; -- Komponentendeklaration
        z: OUT BIT
    );
END COMPONENT;

COMPONENT AND2
PORT (x, y: IN BIT;
         z: OUT BIT
     );
END COMPONENT;

BEGIN
U0: XOR2 PORT MAP (a, b, erg); -- verwendete Komponenten
U1: AND2 PORT MAP (a, b, c); -- Verdrahtung gemäß Reihenfolge
                                in den Listen

END halfadd_arch;
```

Bild 6.14 zeigt den Aufbau eines VHDL-Modells in strukturaler Beschreibung. Dabei sind drei Komponenten eingebunden, die Teil des Modells werden.

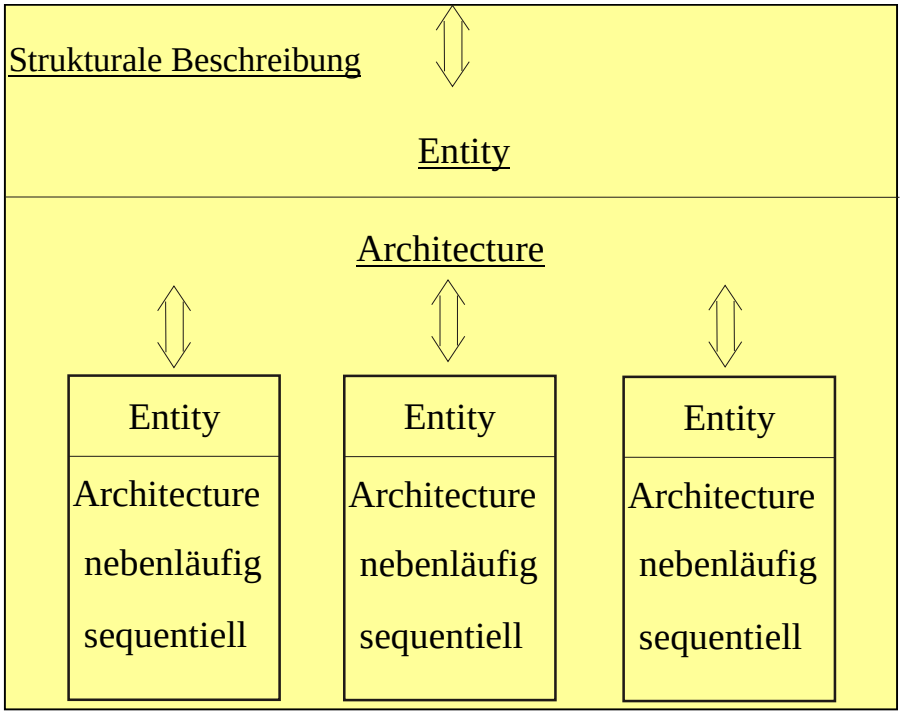


Bild 6.14: Aufbau eines VHDL-Modells aus strukturaler Sicht

6.4.12 CONFIGURATION

In der **CONFIGURATION** wird zunächst festgelegt, welche **ARCHITECTURE** zu verwenden ist. Bei strukturalen Beschreibungen kann zusätzlich angegeben werden, aus welchen Bibliotheken die einzelnen Submodule entnommen werden, wie sie eingesetzt (verdrahtet) werden, und welche Parameter (**GENERICS**) übergeben werden sollen. Zu einer **ENTITY** können mehrere Konfigurationen gehören. Dadurch wird der Entwickler beim Experimentieren mit verschiedenen Varianten unterstützt, weil er unterschiedliche Architekturen und Komponenten in der Konfiguration zusammenstellen kann.

Beispiel:

```
CONFIGURATION signale_config OF signale IS
FOR simulate
        FOR prog: signale USE ENTITY work.signale(signale_arch);
        END FOR;
END FOR;
END signale_config;
```

6.4.13 TESTBENCH (Einführung)

Zu einem vollständigen VHDL-Entwurf gehört auch eine Testumgebung, „Testbench“ genannt. Sie dient der Überprüfung entwickelter Beschreibungen. Eine Testbench bindet den Entwurf als Komponente ein und weist den Eingangssignalen Werte zu. Anhand der Werte der Ausgangssignale kann die Funktion und die Korrektheit des Entwurfs untersucht werden.

Oft ist es erforderlich, die Instruktionen in einer Testbench sequentiell mit einer definierten Geschwindigkeit abzuarbeiten. Um dies zu erreichen, wird der Wait-Befehl verwendet. So können Testbenches entwickelt werden, die bei jedem Takt die Eingangssignale neu belegen.

Die Testbench muß stets als letzte Einheit kompiliert werden, bevor sie in den Simulator geladen werden kann.

Im folgenden Beispiel ist eine Testbench dargestellt, die das Modell ’testdesign’ als Komponente einbindet und sein Eingangssignal ***x*** belegt.

Beispiel:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY testbench IS     -- keine Portdeklaration
END testbench;

ARCHITECTURE simulate OF testbench IS
COMPONENT testdesign
PORT (x: IN  INTEGER;
      y: OUT INTEGER
     );
END COMPONENT;

SIGNAL eingang, ausgang: INTEGER;

BEGIN
  test: testdesign PORT MAP (eingang, ausgang);
PROCESS
BEGIN
  eingang <='1';
  WAIT FOR 20 ns;
  LOOP                    -- Endlosschleife für 25 MHz-Takt
    eingang <='0';
    WAIT FOR 20 ns;
  END LOOP;
END PROCESS;
END simulate;

CONFIGURATION test_simulate OF testbench IS
 FOR simulate
 FOR test: testdesign USE ENTITY work.testdesign(testdesign_arch);
    END FOR;
  END FOR;
END test_simulate;
```

6.4.14 Die Datentypen **std_ulogic** und **std_logic**

Im IEEE-Standard 1164-1993 wird eine neunwertige Logik eingeführt, die die in Tabelle 6.3 angegebenen Werte umfaßt. In der VHDL-Bibliothek
ieee.std_logic_1164 werden diese Werte durch die Deklarationen

```
type std_ulogic IS ('U','X','0','l','Z','W','L','H','-');
type std_logic IS ('U','X','0','l','Z','W','L','H','-');
```

zu den Datentypen **std_ulogic** und **std_logic** zusammengefaßt.

Wert	Bedeutung	Verwendung
'U'	nicht initialisiert	nicht initialisiertes Signal im Simulator
'X'	undefiniert	Simulator erkennt mehr als einen aktiven Signaltreiber ⇒ Buskonflikt liegt vor
'0'	starke logische '0'	entspricht einem auf Null liegenden Signal der Breite 1bit
'1'	starke logische 'l'	entspricht einem auf Eins liegenden Signal der Breite 1bit
'Z'	hochohmig	Tri-State Ausgang
'W'	schwach unbekannt	Simulator erkennt Buskonflikt zwischen 'L'- und 'H'-Pegel
'L'	schwache logische '0'	z.B. Ausgang mit Pull-Down-Widerstand (Open Source)
'H'	schwache logische '1'	z.B. Ausgang mit Pull-Up-Widerstand (Open Drain)
'-'	don't care	Zustand ohne Bedeutung; kann für Minimierung verwendet werden

Tabelle 6.3: Wertevorrat der Standardlogik-Datentypen

Von den Standardlogik-Datentypen sind auch der Typ X01 mit dreiwertiger Logik sowie ein Typ X01Z jeweils mit dem im Namen angegebenen Wertevorrat abgeleitet. Sie stellen Untermengen von **std_(u)logic** dar.

Worin besteht nun der Unterschied zwischen den Datentypen **std_logic** und **std_ulogic**? Für Signale des Typs **std_ulogic** darf immer nur ein Treiber existieren, d.h. solche Signale dürfen nur in einem Prozeß bzw. einer nebenläufigen Anweisung eine Wertzuweisung erfahren, auch wenn die Signalzuweisung im VHDL-Code zu unterschiedlichen Zeitpunkten erfolgt. Zu berücksichtigen ist nämlich, daß alle Treiber ständig Signalwerte liefern. Signalkonflikte, die in der Hardware zu Schaltungsfehlern führen würden, werden bei diesem Datentyp also nicht aufgelöst. Das „**u**" in der Typbezeichnung drückt dies aus, es steht für **unresolved data type**. Ein VHDL-Compiler erkennt solche Zustände und bricht mit einer Fehlermeldung ab.

Beim Datentyp **std_logic** ist das anders. Hier dürfen mehrere Treiber für ein Signal existieren. Dieser Datentyp wird insbesondere bei bidirektionalen Bussen benötigt, auf denen die Daten von mehreren Sendern kommen können. Die Entscheidung, welcher Treiber sich durchsetzt, trifft der Simulator auf Basis der im **1164 IEEE-Standard** definierten Auflösungsfunktion (Resolution Function).

Für den Anwender ist es wichtig zu wissen, wie die Auflösungsfunktion des Datentyps **std_logic** wirkt. Sie ist gemäß der in der in Tabelle 6.4 angegebenen Matrix im **IEEE 1164-Paket** abgelegt. Der resultierende Signalwert, der sich bei gleichzeitiger Aktivität zweier Signaltreiber ergibt, ist dort zu entnehmen. Die aktuellen Signalwerte der beiden gleichzeitig aktiven Treiber

sind jeweils in der ersten Zeile bzw. Spalte einzusetzen. Das Resultat ist im Kreuzungspunkt abzulesen.

	U	X	0	1	Z	W	L	H	-
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
-	U	X	X	X	X	X	X	X	X

Tabelle 6.4: Auflösungsfunktion für den Datentyp **`std_logic`**

(Aus Gründen der Übersicht wurden die Anführungszeichen in der Tabelle weggelassen)

Die Auflösungsfunktion wird im VHDL-Simulator automatisch aktiviert, sobald ein Signal durch mehr als einen Prozeß beeinflußt wird. Das bedeutet, daß versehentliche Mehrfachzuweisungen an ein Signal beim Datentyp **`std_logic`** vom Compiler während der Entwurfsphase nicht beanstandet werden. Tabelle 6.4 ist zu entnehmen, daß ein Signal, das nicht initialisiert (‘U‘) oder undefiniert (‘X‘) ist, beim Zusammenschalten durch kein anderes Signal verändert werden kann.

Da jede boolesche Verknüpfung von ‘U‘ bzw. ‘X‘ mit einem anderen Signalwert wie z.B. ‘1‘ oder ‘0‘ als Resultat stets wieder einen ‘U‘- bzw. ‘X‘-Zustand generiert, muß zunächst immer der undefinierte Zustand beseitigt werden, bevor verwendbare ‘l‘- oder ‘0‘-Pegel zur Verfügung stehen. Ein einfaches Beispiel ist ein Flipflop, wo der D-Eingang mit einem Ausgang (in der Regel $\overline{Q}$) verbunden ist. Da alle Signale zunächst im Simulator mit ‘**U**‘ belegt werden, sind die Flipflop-Ausgänge solange undefiniert, bis ein **Reset** erfolgt ist.

Der undefinierte Zustand ‘X‘ entsteht bei einem Buskonflikt, z.B. wenn gleichzeitig ein Sender eine ‘0‘ und ein anderer eine ‘l‘ auf ein- und dieselbe Busleitung legt. Der hochohmige Zustand ‘Z‘ muß durch einen Tristate Treiber explizit definiert werden. ‘Z‘ wird durch jeden anderen Signalwert überschrieben.

Im VHDL-Quellcode können die Datentypen **`std_ulogic`** und **`std_logic`** verwendet werden, wenn vor der **ENTITY**-Deklaration die IEEE-Bibliothek mit Hilfe einer **LIBRARY**-Anweisung deklariert wurde. Zusätzlich muß durch eine **USE**-Anweisung angeben werden, daß alle Komponenten des **`std_logic_1164`**-Pakets verwendet werden sollen. Die beiden folgenden Zeilen müssen sich demnach vor jeder Entity befinden, in der ein **`std_(u)logic`**-Datentyp benutzt werden soll:

```
library ieee;
use ieee.std_logic_1164.all;
```

Während sich die **LIBRARY**-Anweisung auf alle Einheiten einer VHDL-Datei bezieht, ist bei der **USE**-Anweisung zu beachten, daß diese vor jeder einzelnen **ENTITY**-Deklaration wiederholt werden muß, die entsprechende Datentypen verwendet. Falls in einer VHDL-Datei eine **ARCHITECTURE** angegeben wird, zu der die zugehörige **ENTITY** in einer anderen Datei abgelegt ist, dann müssen vor dieser **ARCHITECTURE** sowohl die **LIBRARY**- als auch die **USE**-Anweisung plaziert werden.

6.5 Simulation von VHDL-Entwürfen (Gebrauch von Testbenches)

Im folgenden wird eine Methodik entwickelt, mit der VHDL-Entwürfe schnell und einfach syntaktisch und funktional überprüft werden können. Da das Erlernen von VHDL wird deutlich beschleunigt, wenn eigene Entwürfe sofort auf Korrektheit überprüft und durchsimuliert werden können.

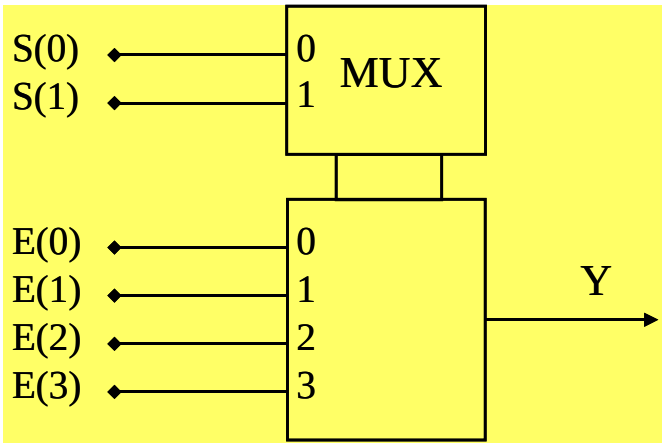
Es wird davon ausgegangen, daß keine komfortable (spezialisierte) Umgebung zur interaktiven Eingabe von Anregungssignalen (Stimuli) zur Verfügung steht, sondern es wird der prinzipielle Aufbau und die Nutzung von Testbenches erläutert. Im allgemeinen können alle benötigten Entwurfseinheiten - also auch die Testbench selbst - zusammen in einer Datei mit der Erweiterung ***.VHD** abgespeichert werden. Eine bessere Übersicht erhält man jedoch, wenn die einzelnen **ENTITY/ARCHITECTURE**-Paare jeweils in einer eigenen Datei abgelegt werden. Nach erfolgreichem Compilieren der Datei, die das zu untersuchende Modell enthält, wird der resultierende Objektcode in einer Bibliothek mit dem vordefinierten Namen WORK abgelegt. Anschließend muß eine Datei angelegt und compiliert werden, die die **Entity** und **Architecture** der Testbench enthält.

Diese TESTBENCH-ENTITY enthält keinerlei Schnittstellensignale. Die Schnittstellensignale des zu untersuchenden Modells werden vielmehr als lokale Signale in der TESTBENCH-ARCHITECTURE deklariert. Diese enthält eine Komponente, die der zu untersuchenden Entity entspricht und an die die lokalen Testbenchsignale übergeben werden. Die Definition des Zeitverlaufs der Stimuli erfolgt durch nebenläufige Anweisungen, in denen die Zeitpunkte der Signalübergänge festgelegt sind. Der syntaktische Aufbau einer TESTBENCH sieht wie folgt aus:

```
entity TEST is

end TEST;

architecture TESTBENCH of TEST is

signal      <Signalliste>;              -- alle Schnittstellensignale der
                                        -- zu untersuchenden entity

component <Komponentenname>             -- identisch mit Entityname der
                                        -- zu untersuchenden entity

            <Port-Liste>                -- identisch mit der Port-Liste
                                        -- der zu untersuchenden entity

end component;

for all:    <Komponentenname>  use  entity  work.<Entityname>
            (<Architekturname>);

begin
            -- nebenläufige Signalzuweisungen an die Stimuli

            -- mit Angabe des Zeitpunkts der Signalübergänge

[<Bezeichner>]: <Komponentenname>

port map (< Liste der angeschlossenen aktuellen Signale>);

end TESTBENCH;
```

Testbencherstellung am Beispiel eines 4-zu-1-Multiplexers

```
entity MUX4X1 is
port (    S: in bit_vector(1 downto 0); -- 2 Select-Eingänge S(0), S(1)
          E: in bit_vector(3 downto 0); -- 4bit Eingangsdaten E
          Y: out bit);                  --  1-bit-Ausgang Y
end MUX4X1;
architecture VERHALTEN of MUX4X1 is
begin
with S select -- Auswahlsignal
          Y <=      E(0) when "00",
                    E(1) when "01",
                    E(2) when "10",
                    E(3) when "11";
end VERHALTEN;
```



Die in der folgenden TESTBENCH deklarierten Signale S1, E1 und Y1 werden als aktuelle Signale an die als Komponente **C1** plazierte Entity MUX4X1 übergeben. Zum Zeitpunkt *t*=0 wird der Eingangsvektor E1 mit "1010" und zum Zeitpunkt *t*=400ns mit "0101" vorbelegt. Das Auswahlsignal S1 durchläuft mit einer Impulsdauer von 100ns zweimal die Sequenzen "00", "01", "10" und "11", womit sich eine Simulationsdauer von 800ns ergibt.

```
-- Testbench für 4-zu-1-Multiplexer
entity TEST is
end TEST;
architecture VERHALTEN of TEST is
signal S1: bit_vector(1 downto 0);
signal E1: bit_vector(3 downto 0);
signal Y1: bit;
component MUX4X1 is
          port(     S: in bit_vector(1 downto 0);
                    E: in bit_vector(3 downto 0);
                    Y: out bit);
end component;
```

```
for all: MUX4X1
use entity work.MUX4X1(VERHALTEN); -- Modell aus der Bibliothek "work"
    begin
        E1 <= "1010", "0101" after 400 ns;
        S1 <= "00", "01" after 100 ns, "10" after 200 ns,
        "11", after 300 ns, "00" after 400 ns, "01" after 500 ns,
        "10" after 600 ns, "11" after 700 ns;
C1:     MUX4X1 port map(S1, E1, Y1); -- Anschließen der Signale
end VERHALTEN;
```

Bei Verwendung der Zeiteinheiten ist zu beachten, daß zwischen dem Zahlenwert und der Einheit ein Leerzeichen stehen muß.

Der wesentliche Vorteil einer rein VHDL-basierten Testumgebung – wie sie hier vorgestellt wurde - besteht in der Unabhängigkeit von der Kommandosprache eines speziellen Simulators und in der einheitlichen Beschreibung von Testobjekt und Quellen der Anregungssignale (Stimuli). Eine entscheidende Qualitätssteigerung für den Test komplexer Entwürfe entsteht dadurch, daß die Testumgebung um Funktionen erweitert werden kann, die ergänzend zu den Stimuli einen Soll-Ist-Vergleich der Ausgangssignale durchführen.

Im folgenden Beispiel werden Varianten der Zuweisung bzw. Abfrage von **bit_vector**-Konstanten in einer Wahrheitstabelle vorgestellt. Dazu dient ein Impulsmustergenerator mit vier Ausgangskanälen **A**, der 8 Signalmuster erzeugen kann, die durch einen 3 bit breiten, Eingangssignalvektor **E** ausgewählt werden. Die Muster sind in Tabelle 6.5 dargestellt. Im VHDL-Code sieht man, daß dem Bit-String stets die Basis des gewählten Zahlensystems voranzustellen ist.

```
                                    -- Vierkanal-Impulsgenerator auf Basis einer Wertetabelle
entity IMPULSGEN is
port (E: in bit_vector(2 downto 0);
      A: out bit_vector(3 downto 0));
end IMPULSGEN;

architecture W_TAB of IMPULSGEN is
begin
P1: process (E)  begin
                 case E is
                 when o"0" => A <= x"7";
                 when o"1" => A <= x"A";
                 when o"2" => A <= x"3";
                 when o"3" => A <= x"F";
                 when o"4" => A <= x"6";
                 when o"5" => A <= x"C";
                 when o"6" => A <= x"0";
                 when o"7" => A <= x"E";
end case;
end process Pl;
end W_TAB;
```

E	A
0	7h
1	Ah
2	3h
3	Fh
4	6h
5	Ch
6	0h
7	Eh

Tabelle 6.5: 8 Signalmuster

Die Buchstaben zur Kennzeichnung von Zahlenbasen sind:

Basis	Buchstabe	Beispiel
Dual	B oder b	B“1010_1100“
Oktal	O oder o	o“1_4_7”
HEX	X oder x	x“AF_fe”

6.5.1 Simulationsspezifische Prozesse für Testumgebungen

Mit nicht synthesefähigen Prozessen kann der Entwurf von Testumgebungen vereinfacht werden. Sie ergänzen eine zu synthetisierende **ARCHITECTURE** um spezielle Stimuli-Prozesse. Dabei unterscheidet man:

- Prozesse, die diskrete Stimuli beschreiben
- Prozesse, die periodische Stimuli beschreiben

Bei der vorgestellten Methode müssen die Eingangssignale der zu synthetisierenden **Entity** während der Testphase aus der **port**-Anweisung entfernt werden und als lokale Signale deklariert sein. Nach Abschluß des Tests müssen die Erweiterungen wieder entfernt werden.

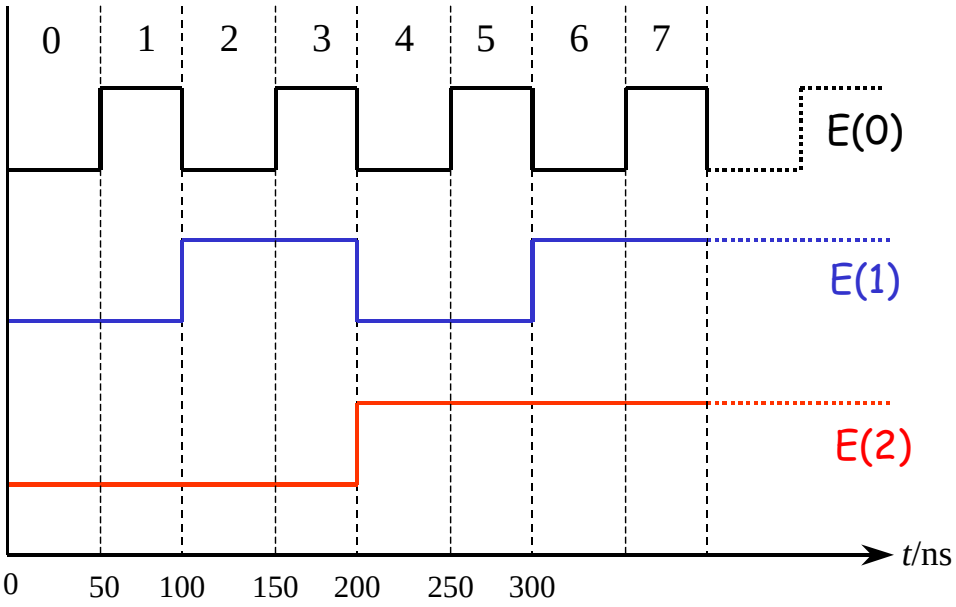
Als Beispiel wird der obige Impulsmustergenerator verwendet.

```
-- Vierkanal-Impulsgenerator mit Testbench

entity  IMPULSGEN is

port (A: out bit_vector(3 downto 0));

end IMPULSGEN;


architecture W_TAB of IMPULSGEN is

signal E: bit_vector(2 downto 0); -- lokales Signal nur für Testbench

begin

-- zu synthetisierender Prozeß
Pl: process (E)
                begin
                case E is
                when o"0" => A <= x"7";
                when o"1" => A <= x"A";
                when o"2" => A <= x"3";
                when o"3" => A <= x"F";
                when o"4" => A <= x"6";
                when o"5" => A <= x"C";
                when o"6" => A <= x"0";
                when o"7" => A <= x"E";
end case;
end process Pl;
```

```
                                -- nicht synthetisierbare Testbench-Prozesse
STIMULI: process                -- nichtperiodische Stimuli
        begin
        E(1) <= '0', '1' after 100 ns, '0' after 200 ns, '1' after 300 ns;
        E(2) <='0', '1' after 200 ns;
        wait;                   -- keine weiteren Signaländerungen
end process STIMULI;
TAKTGEN: process
            begin               -- periodischer Stimulus
            E(0) <= '0';
            wait for 50 ns;
            E(0) <= '1';
            wait for 50 ns;
end process TAKTGEN;
-- Ende der Testbench
end W_TAB;
```

Aufgabe der Testumgebung ist es, alle möglichen Kombinationen des Eingangsvektors **E** zu generieren. Dafür wird das Eingangssignal **E** aus der ursprünglichen **entity** entfernt und lokal deklariert. Dem niederwertigsten Bit E(0) wird in dem Prozeß TAKTGEN ein mit 50 ns periodischer Stimulus zugewiesen. Da der ohne Empfindlichkeitsliste formulierte Prozeß nach Ablauf der zweiten Wartephase erneut mit der Ausführung beginnt, wird ein Signal mit einer Periodendauer von 100 ns und einem Tastverhältnis von 1:1 erzeugt.

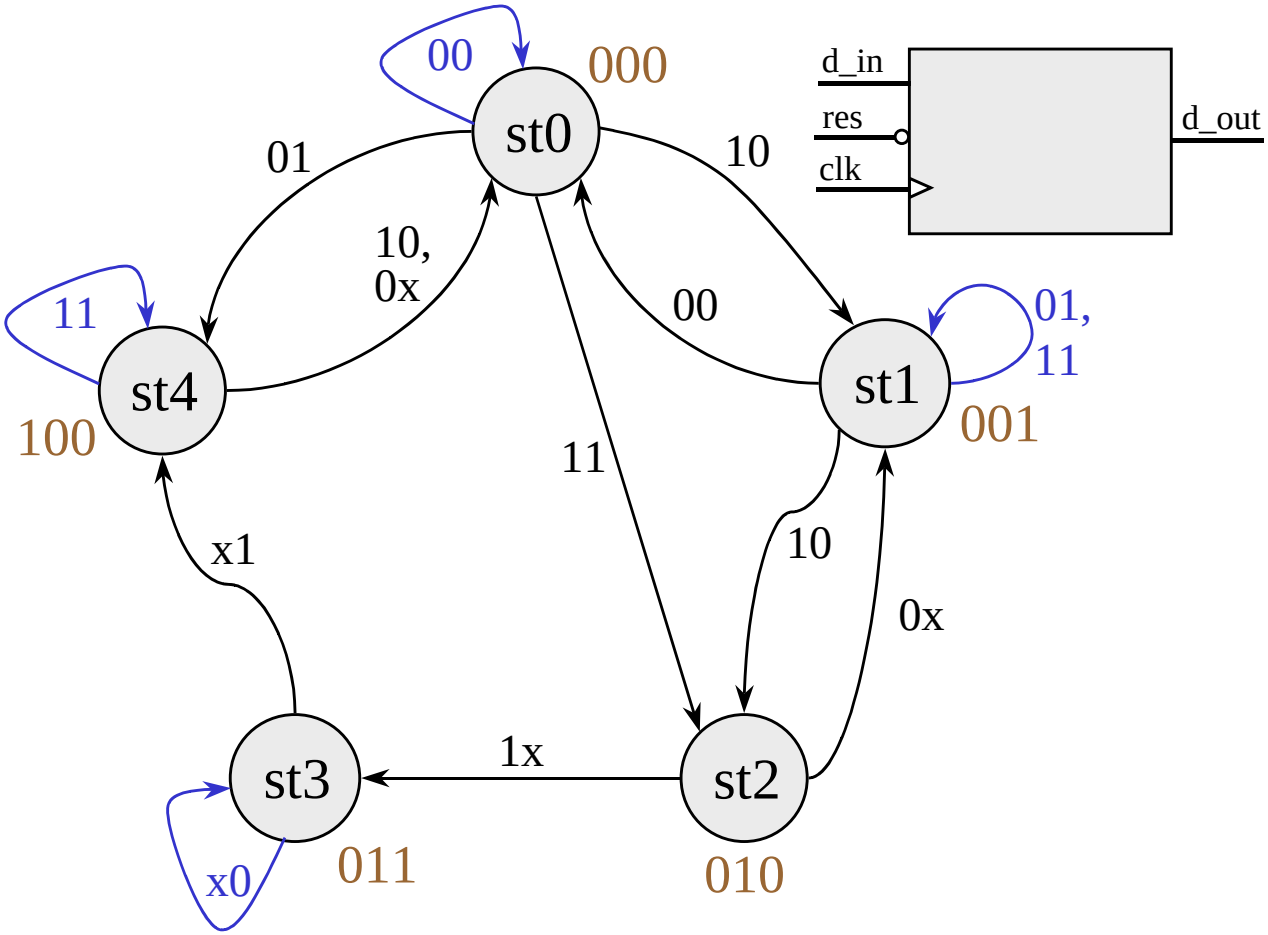
Die Signalwechsel der beiden anderen Bits E(l) und E(2) werden durch verzögerte, unbedingte Signalzuweisungen innerhalb des Prozesses STIMULI realisiert. Wie das Beispiel zeigt, ist es möglich, mehrere Signalübergänge in einer einzigen unbedingten Signalzuweisung durch Verwendung des Schlüsselworts **after** zu definieren. Wichtig ist, daß alle Übergänge eines Signals in ***einer*** Anweisung stehen, da eine zweite Anweisung zu einem ***Signaltreiberkonflikt*** führen würde. Dieser Prozeß endet mit einer unbedingten **wait**-Anweisung.



Die Signale E(l) und E(2) hätten bei den gewählten Signalmustern auch durch je einen weiteren Taktgeneratorprozeß mit einer Periode von 100 ns bzw. 200 ns definiert werden können.

6.5.1.1 Vergleich von VHDL und VERILOG an einem FSM-Beispiel

Es wird die FSM-Beschreibung des MOORE-Automaten nach Bild 6.13 verwendet.



Modellierung in VERILOG

```
module M_AUT (d_in, d_out, res, clk);
output [2:0] d_out;
input [1:0] d_in;
input res, clk;
reg d_out;
reg [2:0] akt_zust, folg_zust;
parameter st0=3'd0, st1=3'd1, st2=3'd2, st3=3d'3, st4=3'd4;

//FSM-Register-Definition
always @ (posedge clk or negedge res)
        begin: Zust_Reg
        if (!res)
        akt_zust = st0;
else
        akt_zust = folg_zust;
end // Zust_Reg

// Kombinatorik der FSM
```

```
always @ (akt_zust or d_in)
        begin: FSM
                case (akt_zust)
                st0:    case (d_in)
                    2'b00:      folg_zust=st0;
                    2'b01:      folg_zust=st4;
                    2'b10:      folg_zust=st1;
                    2'b11:      folg_zust=st2
endcase
                stl:    case (d_in)
                    2'b00:      folg_zust=st0;
                    2'b10:      folg_zust=st2;
                    default:    folg_zust=st1;
endcase
                st2:    case (d_in)
                    2'b0x:      folg_zust=st1;
                    2'b1x:      folg_zust=st3;
endcase
                st3:    case (d_in)
                    2'bx1:      folg_zust=st4;
                    default:    folg_zust=st3;
endcase
                st4:    case (d_in)
                    2'b11:      folg_zust=st4;
                    default:    folg_zust=st0;
endcase
default:                        folg_zust=st0;
endcase
end // FSM

// Moore-Ausgabe (hängt nur vom aktuellen Zustand ab)
always @ (akt_zust)
begin: Ausgabe
                case (akt_zust)
                st0:        d_out = 3'b000;
                st1:        d_out = 3'b001;
                st2:        d_out = 3'b010;
                st3:        d_out = 3'b011;
                st4:        d_out = 3'bl00;
default:                    d_out = 3'b000;
endcase
end // Ausgabe
endmodule // M_AUT
```

6.5.1.2 VHDL–Namenskonventionen und reservierte Bezeichner

- VHDL unterscheidet nicht zwischen Groß- und Kleinschreibung
- Zwei Gedankenstriche -- leiten eine Kommentarzeile ein
- Namen können alphanumerische Zeichen und den Unterstrich _ beinhalten
- Namen müssen mit einem Buchstaben anfangen
- Es dürfen keine zwei Unterstriche __ in einer Zeile benutzen, oder ein Unterstrich als letztes Zeichen eines Namens vorkommen
- Zwischenräume sind in Namen nicht erlaubt
- Objektnamen müssen eindeutig sein. So darf z.B. kein Signal mit dem Namen **A** und gleichzeitig ein Bus mit der Bezeichnung **A(7 downto 0)** vorkommen

Im folgenden sind die in VHDL reservierten Schlüsselwörter aufgelistet:

abs	downto	library	postponed	subtype
access	else	linkage	procedure	then
after	elsif	literal	process	to
alias	end	loop	pure	transport
all	entity	map	range	type
and	exit	mod	record	unaffected
architecture	file	nand	register	units
array	for	new	reject	until
assert	function	next	rem	use
attribute	generate	nor	report	variable
begin	generic	not	return	wait
block	group	null	rol	when
body	guarded	of	ror	while
buffer	if	on	select	with
component	inertial	others	signal	
bus	impure	open	severity	xnor
case	in	or	shared	xor
configuration	inout	out	sla	
constant	is	package	sra	
disconnect	label	port	srl	

Tabelle 6.6: VHDL-Schlüsselwörter

6.5.1.3 Verilog–Namenskonventionen und reservierte Bezeichner

- In Verilog wird zwischen Groß- und Kleinschreibung unterschieden
- Zwei Schrägstriche **//** leiten eine Kommentarzeile ein
- Ein Schrägstrich und ein Stern **/*** leiten mehrzeilige Kommentare ein
- Ein Stern gefolgt von einem Schrägstrich und ein ***/** beenden mehrzeilige Kommentare
- Namen können alphanumerische Zeichen, den Unterstrich **_** und das $-Zeichen beinhalten
- Namen müssen mit einem Buchstaben anfangen oder mit **_**
- Zwischenräume sind in Namen nicht erlaubt

Im folgenden sind die in VERILOG reservierten Schlüsselwörter aufgelistet:

always	endmodule	medium	reg	tranif0
and	endprimitive	module	release	tranif1
assign	endspecify	nand	repeat	tri
attribute	endtable	negedge	rnmos	tri0
begin	endtask	nmos	rpmos	tri1
buf	event	nor	rtran	triand
bufif0	for	not	rtranif0	trior
bufif1	force	notif0	rtranif1	trireg
case	forever	notif1	scalared	unsigned
casex	fork	or	signed	vectored
casez	function	output	small	wait
cmos	highz0	parameter	specify	wand
deassign	highz1	pmos	specparam	weak0
default	if	posedge	strength	weak1
defparam	ifnone	primitive	strong0	while
disable	initial	pull0	strong1	wire
edge	inout	pull1	supply0	wor
else	input	pulldown	supply1	xnor
end	integer	pullup	table	xor
endattribute	join	remos	task	
endcase	large	real	time	
endfunction	macromodule	realtime	tran	

Tabelle 6.7: Schlüsselwörter für Verilog

6.5.2 Bewertung von VHDL

Der Einsatz einer standardisierten Hardwarebeschreibungssprache bietet klare Vorteile für den Entwurf komplexer elektronischer Schaltungen, aber es gibt leider auch Nachteile. So kann z.B. die umfangreiche Syntax zwar viele Modellierungsmöglichkeiten bieten, hat aber den Nachteil eines hohen Einarbeitungsaufwands.

Vorteile:

- Vielseitigkeit

VHDL ist eine Sprache für viele Zwecke, die sowohl spezifikations- und simulationsgeeignet ist und auch als Ein- und Ausgabesprache für die Schaltungssynthese verwendet werde kann. Durch die firmenunabhängige Standardisierung ist ein Datenaustausch zwischen

- verschiedenen Programmen
- verschiedenen Entwurfsebenen,
- verschiedenen Projektteams
- sowie Entwickler und Hersteller

möglich.

- Programmunabhängigkeit

Neben VHDL gibt es auch noch andere Hardwarebeschreibungssprachen und Datenformate. Man unterscheidet dabei zwischen programmspezifischen und nicht programmspezifischen Sprachen bzw. Formaten. Erstere sind an einen bestimmten Software-Hersteller gebunden und werden meist nicht von den Programmen anderer Hersteller unterstützt. Auch die heutige Hauptkonkurrenz von VHDL, **Verilog-HDL**, war lange Zeit herstellerspezifisch und hat sich erst im Verlauf der letzten Jahre zu einer unabhängigen HDL entwickelt. VHDL ist vom Ansatz her programmunabhängig. Es gibt eine Vielzahl von Software-Anbietern, die für viele Entwurfsschritte integrierter Schaltungen VHDL-Lösungen anbieten. Man kann somit unter mehreren Alternativen die zur Lösung der anstehenden Aufgabe optimale Software aus wählen. Bei VHDL stand von Anfang an die Unabhängigkeit von Rechnersystemen im Vordergrund. Falls dennoch systemabhängige Aspekte erforderliche sind, können sie in **PACKAGES** gekapselt werden, so daß das VHDL-Modell selbst unabhängig vom eingesetzten Rechnersystem bleibt.

- Technologieunabhängigkeit

Die Entscheidung für eine bestimmte Technologie (FPGA, Gate-Array, Standardzelle) muß erst relativ spät getroffen werden. Selbst ein danach eventuell nötiges Umschwenken verursacht kein komplettes Redesign. Durch die Freiheit zur

- Definition von eigenen Logiktypen mit entsprechenden Operatoren
- technologiespezifischen Signalverknüpfungen (wired-or, wired-and)
- neuen, bei strukturalen Modellen eingesetzten Komponentenbibliotheken

können die spezifischen Eigenschaften aller Technologien mit VHDL abgebildet werden.

- Modellierungsmöglichkeiten

VHDL stellt zahlreiche Konstrukte zur Beschreibung von Schaltungen und Systemen zur Verfügung. Damit lassen sich Modelle z.B. auf algorithmischer, Register-Transfer und Logikebene erstellen. Die Modelle können dabei Unterkomponenten verschiedener Entwurfssichten und –ebenen enthalten. Bei der strukturalen Modellierung kann eine mehrstufige Hierarchie implementiert werden. Beschreibungen auf abstraktem Niveau haben mehrere Vorteile: Sie sind kompakt und über-

schaubar. Sie ermöglichen kürzere Entwicklungszeiten, benötigen weniger Rechenzeit bei der Simulation und erlauben eine frühzeitige Verifikation.

- Entwurf komplexer Schaltungen

VHDL hat die Verbreitung von Synthesewerkzeugen verstärkt und so eine neue und produktivere Entwurfsmethodik mit strukturierter Top-Down-Vorgehensweise herbeigeführt. Wesentliche Aspekte dabei sind:

- frühzeitige Entwurfsüberprüfung, da die Spezifikation eine simulierbare Beschreibung darstellt
- Verhaltensbeschreibungen können synthetisiert werden
- zahlreiche Sprachkonstrukte zur Parametrisierung von Modellen erlauben ein einfaches Design von Varianten
- die Entwicklung wieder verwendbarer Modelle wird unterstützt
- rasche Umsetzungsmöglichkeiten auf verschiedene Technologien sind gegeben

- Selbstdokumentation

VHDL ist direkt vom Menschen lesbar. Die Syntax ist sehr ausführlich und hat selbsterklärende Befehle. Wählt man geeignete Objektnamen, so enthält eine Beschreibung genügend Informationen, um zu einem späteren Zeitpunkt problemlos ohne zusätzliche Kommentare interpretiert werden zu können.

Nachteile:

Mit der Verwendung von VHDL beim IC-Entwurf ist weit mehr verbunden als mit der Nutzung irgendeiner neuen Programmiersprache oder eines neuen Formates. Der gesamte Entwurfsablauf hat sich von der früheren manuellen Vorgehensweise mit Schaltplaneingabe auf Logikebene hin zu einer Schaltungsbeschreibung auf RT-Ebene mit anschließender Synthese gewandelt. Dies hat mehrere Folgen:

- Es ist ein grundsätzlich neuer Entwurfsstil nötig, der mit einem Umdenken bei den Hardwareentwicklern verbunden ist. Erfahrungsgemäß haben gerade Hardwareentwickler, die in ihrer Ausbildung oft zu wenig mit Programmiersprachen und der erforderlichen strukturierten Vorgehensweise vertraut gemacht wurden, dabei große Probleme.
- Die erforderlichen Aus- und Weiterbildungsmaßnahmen verursachen Kosten und Ausfallzeiten. Hinzu kommt meist die Neuanschaffung von Rechnern mit genügend Leistung sowie Software-Lizenzen, deren Kosten die der Rechner häufig weit übersteigen.

- Modellierung analoger Systeme

VHDL wurde zunächst mit umfangreichen Beschreibungsmitteln für Digitaltechnik ausgestattet. Im Laufe der Zeit wurden auch Konstrukte zur Modellierung analoger Systeme entwickelt, die auch Komponenten mit mechanischen, optischen, thermischen, akustischen oder hydraulischen Eigenschaften und Funktionen einschließen. Damit ist VHDL auf dem Weg, eine vollständige Verhaltensmodellierung technischer Systeme zu ermöglichen. Die Definition einer erweiterten Norm IEEE 1076.1, AHDL, soll speziell die Modellierung analoger elektronische Schaltungen mit wert- und zeitkontinuierlichen Signalen gestatten. Von der allgemeinen Verwendbarkeit zur Schaltungssynthese – wie bei Digitalschaltungen – ist man jedoch leider noch weit entfernt.

- Komplexität

Obwohl die Komplexität von VHDL aufgrund der vielen Modellierungsmöglichkeiten generell sicher ein Vorteil ist, kann sie sich im Einzelnen aber auch nachteilig auswirken:

- VHDL erfordert einen hohen Einarbeitungsaufwand. Es ist mit einer weitaus längeren Einarbeitungszeit als bei jeder anderen Programmiersprache zu rechnen. Insbesondere muß ein Modellierungsstil ***eingeübt*** werden.
- Das Verhalten eines komplexen VHDL–Modells in der Simulation ist für den Neuling kaum nachvollziehbar, da die zugehörigen Mechanismen nicht von gängigen Programmiersprachen abgeleitet werden können.
- Die Semantik wurde ursprünglich an vielen Stellen nicht eindeutig und klar festgelegt. In der Überarbeitung (1993) wurden einige Unklarheiten beseitigt. Trotzdem bleibt das Nachschlagewerk für VHDL, das Language Reference Manual (LRM) eines der am schwersten zu lesenden Bücher der Welt.

- Synthese–Subsets

VHDL ist als Sprache bezüglich der Syntax und Simulationssemantik standardisiert, nicht jedoch für die Anwendung als Eingabe für Synthesewerkzeuge. Außerdem enthält VHDL Konstrukte, die sich ***prinzipiell nicht*** in eine Hardware umsetzen lassen. Darüber hinaus unterstützt jedes Synthesewerkzeug einen etwas anderen VHDL–Sprachumfang (**Subset**) und erfordert einen spezifischen, angepaßten Modellierungsstil. VHDL–Modelle müssen somit meist auf ein spezielles Synthesewerkzeug zugeschnitten sein. Dies verhindert einen unkomplizierten Wechsel des Werkzeugs und erhöht die Abhängigkeit vom Werkzeughersteller. Der Entwickler muß die Anforderungen des gewählten Werkzeuges kennen und von Anfang an bei der Modellerstellung berücksichtigen.

- Ausführlichkeit

Die Ausführlichkeit von VHDL kann ein Nachteil sein, weil manchmal lange und umständliche Beschreibungen entstehen, so daß allein schon der Umfang des einzugebenden Textes bei der Modellerstellung mittels eines Texteditor ein schnelles Vorgehen verhindert.

6.6 ASIC Technologien und ’Entwurfsstile’

In diesem Abschnitt wird ein kurzer anschaulicher Abriß über die in Bild 6.1 schon vorgestellten ASIC-Technologien und die dazugehörigen Entwurfsstile gegeben. Der Schwerpunkt wird dabei auf den FPGAs liegen, da sie aufgrund ihrer hohen Leistungsfähigkeit und der schnellen und einfachen Konfigurierbarkeit für die verschiedenen Arbeitsfelder des Ingenieurs große Bedeutung haben. Es ist davon auszugehen, daß diese Bedeutung wächst und daß künftig nahezu jeder Ingenieur für sein Arbeitsgebiet den Entwurf anwendungsspezifischer Lösungen beherrschen oder zumindest verstehen muß. Nicht nur der „Digitaltechniker“ und Rechnerspezialist, sondern auch der Kommunikations- Automatisierungs- und der Energietechniker wird sich der neuen Möglichkeiten bedienen, die es weitaus besser gestatten, problemangepaßte , leistungsfähige und preisgünstige Lösungen zu schaffen als dies etwa mit Standardbausteinen auf Leiterplatten jemals möglich wäre.

6.6.1 Vollkundenschaltung (Full Custom IC)

Der Entwurf einer Vollkundenschaltung stellt zum einen hohe Ansprüche an den Entwickler und ist zum anderen der teuerste Weg zur anwendungsspezifischen Schaltung. Der Vorteil ist ein flächen- und leistungsoptimiertes Ergebnis, das in hohen Stückzahlen produziert werden kann und mögliche Konkurrenz lange Zeit aus dem Feld schlägt.

Der vollkundenspezifische Entwurf bietet die meisten Freiheitsgrade. Alle Strukturen werden einzeln ausgelegt und können bezüglich der gegebenen Randbedingungen optimiert werden. D.h. es wird die Dimensionierung und Anordnung der einzelnen Transistoren sowie der Leiterbahnen im Detail durchgeführt.

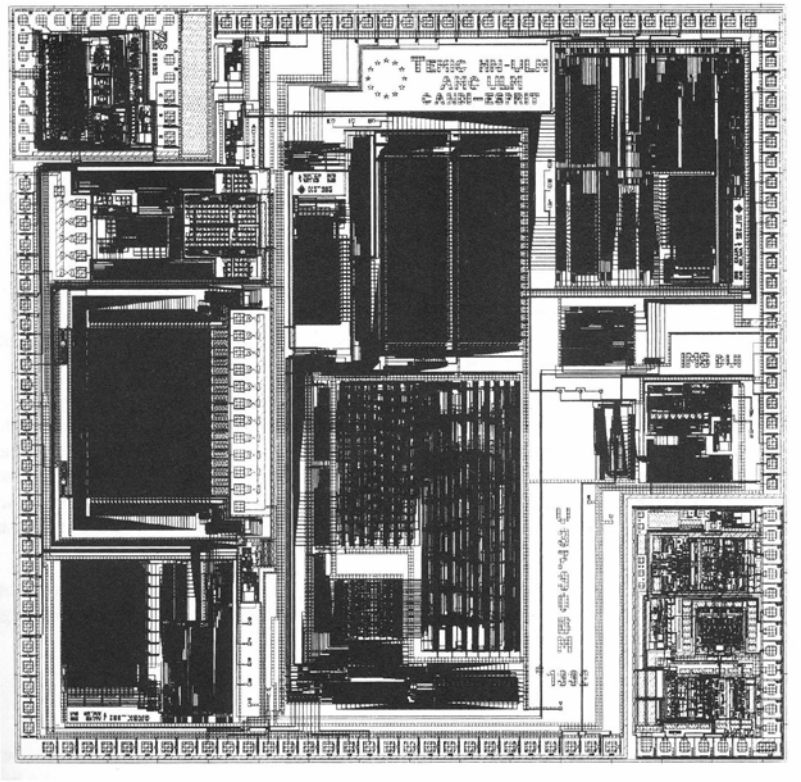


Bild 6.15: Chipbild einer Vollkundenschaltung

Dabei ist die Verwendung beliebiger geometrischer Formen erlaubt, also auch nicht geradlinig begrenzter Strukturen. Diese Freiheitsgrade erlauben die Ausnutzung aller Möglichkeiten, die eine Technologie zu bieten hat.

Die Charakteristika des vollkundenspezifischen Entwurfs im Überblick:

- hohes Optimierungspotential

  Da jede einzelne geometrische Struktur explizit zu entwerfen ist, kann sie den jeweils erforderlichen Gegebenheiten optimal angepaßt werden. Insbesondere im Hinblick auf höchste Packungsdichte, d.h. optimale Siliziumausnutzung, ist dies von Interesse. Voraussetzung ist die Verfügbarkeit von technologisch geschultem Personal.

- hohe Technologietreue

  Die detaillierte Realisierung der einzelnen Strukturen ermöglicht die Berücksichtigung komplexer technologischer Randbedingungen. In der Praxis können so einzelne „Design Rules“ gezielt ignoriert werden, was bei standardisierten (worst-case-orientierten) Verfahren unmöglich wäre.

- hohes Fehlerrisiko

  Wegen der Vielzahl der zu behandelnden Strukturen, die eine Vielfalt von Randbedingungen, mit teilweise komplizierten Wechselwirkungseffekten, einzuhalten haben, ist klar, daß damit ein hohes Fehlerrisiko verbunden ist. Eine rein manuelle Sicherstellung eines korrekten Entwurfs wird generell unmöglich sein, schon gar nicht durch visuelle Überprüfung eines fertigen Layouts, z.B. auf Einhaltung aller geometrischen Randbedingungen.

- hoher Erstellungs- und Korrekturaufwand

  Die Behandlung jeder einzelnen Struktur eines Entwurfs führt dazu, daß bei größeren Objekten ein hoher Zeitaufwand für Anordnung und Optimierung erforderlich wird. Hinzu kommt, daß beim Auftreten von Fehlern, weiterer Zeitaufwand für Korrekturen anfällt. Gewöhnlich führen Fehler zu teuren 'Redesigns', weil man oft bis in sehr frühe Entwurfsstadien zurückgehen muß.

Die obigen Betrachtungen zeigen, daß vollkundenspezifischer Entwurf nur in wenigen Anwendungsbereichen sinnvoll ist, z.B. da wo analoge oder gemischt analog/digitale Schaltungen aufgrund ihrer Zielspezifikation keine standardisierten Verfahren zulassen, oder wo höchste Packungsdichte zur Realisierung der Funktionalität erforderlich ist. Des weiteren ist vollkundenspezifischer Entwurf vertretbar, wenn die zu erwartenden Stückzahlen (ein Grenze kann heute bei etwa mehreren Millionen Stück pro Jahr gezogen werden) hohe Entwurfskosten eine lange Entwicklungszeit rechtfertigen; das wäre z.B. bei universell einsetzbaren programmierbaren Standardschaltungen wie Mikroprozessoren oder Mikrocontrollern und auch DSPs gegeben.

6.6.2 Standardzellenentwurf (Cell-Arrays)

Beim Standardzellenentwurf müssen ebenso wie bei einer Vollkundenschaltung alle Schritte der Halbeiterherstellung durchlaufen werden. Dabei entstehen – in erster Linie aufgrund der Maskenfertigung - hohe Kosten, bevor ICs zum Testen zur Verfügung stehen. Das Charakteristikum des Standardzellenentwurfs sind Einschränkungen der Freiheitsgrade durch Vorgabe standardisierter Funktionseinheiten, wie sie häufig bei digitalen Schaltungen vorkommen. Man kann auf vorentworfene Grundlayouts zurückgreifen, die in „Zellbibliotheken" bereitgestellt werden. Die Layouts dieser Zellen sind hinsichtlich funktionaler Korrektheit und der elektrischen Eigenschaften verifiziert. Sie können aus den Bibliotheken wie aus einem Baukasten entnommen und auf dem Silizium nach einfachen Regeln zusammengesetzt werden.

Anhand von Bild 6.16 lassen sich die wesentlichen Merkmale des Standardzellenentwurfes beschreiben. Man erkennt eine ausgeprägte Zeilenstruktur, die sich dadurch ergibt, daß für das Layout aller Bibliothekszellen immer rechteckige Zellen identischer Höhe verwendet werden, die an zwei gegenüberliegenden Seiten über festgelegte Positionen für Versorgungsleitungen und Ein- bzw. Ausgangsanschlüsse verfügen. An den beiden verbleibenden Zellseiten ist eine freie Anordnung von Anschlüssen zugelassen.

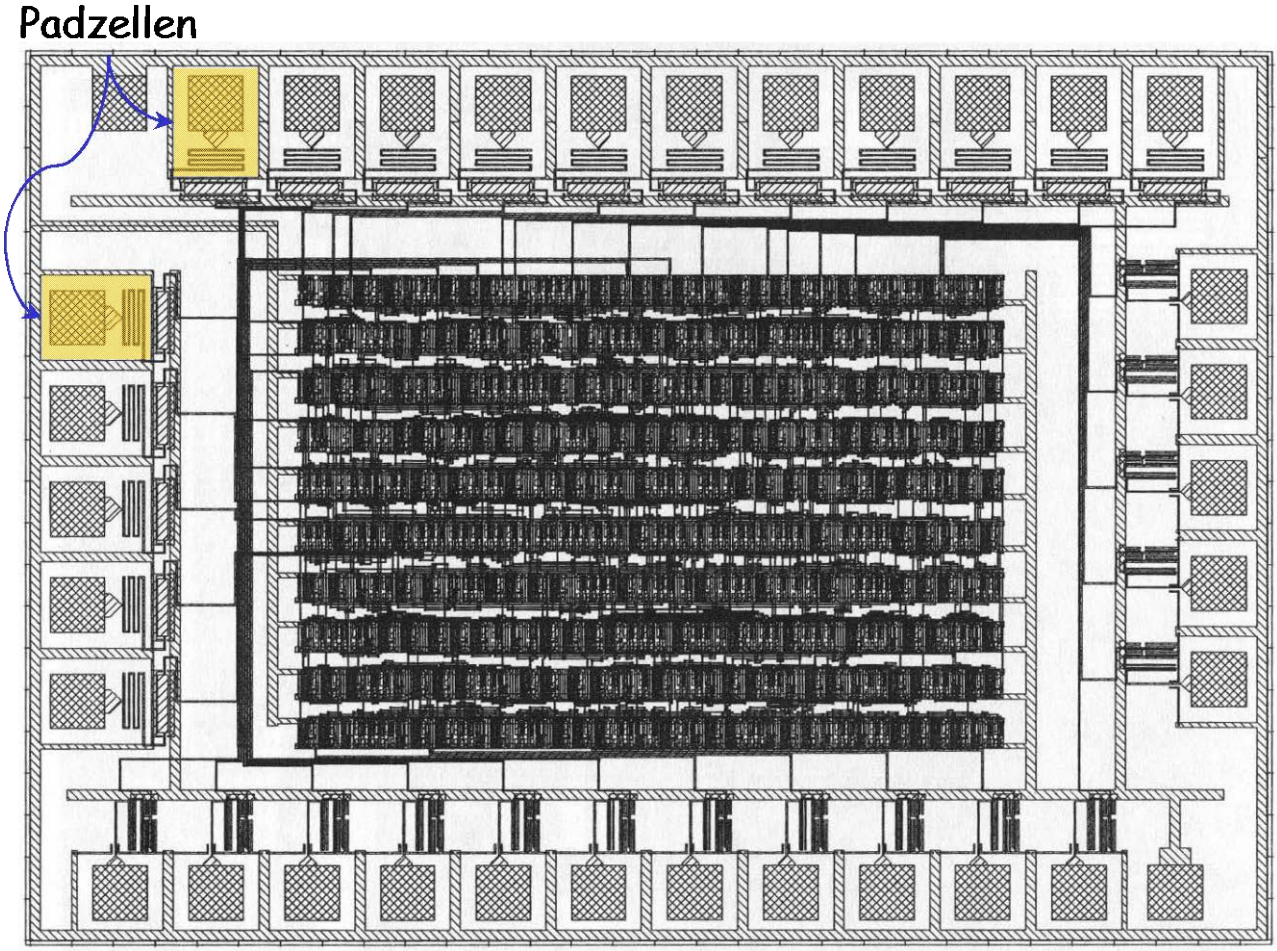


Bild 6.16: Chipbild eines Standardzellen–ICs

Diese Art der Standardisierung der Zellen ermöglicht es, komplexe Funktionen aufzubauen, indem Reihen von Zellen durch Aneinanderfügen, ohne zusätzliche Verbindungsstrukturen, gebildet werden. Der Entwurf von Schaltungen aus Standardzellen besteht somit aus der Auswahl von Zellen, die in Reihen angeordnet werden, der Plazierung der einzelnen Reihen in vorgegebenen Rasterabständen auf der Chipoberfläche und der Verdrahtung.

Der Standardzellentwurf ist folgendermaßen charakterisiert:

- einfache Zellanordnung

  Die Zahl ’Design Rules’ ist gegenüber dem vollkundenspezifischen Entwurf drastisch reduziert. Es sind praktisch nur noch Regeln zu beachten, die den Zellrand und die Verdrahtung betreffen. Aufgrund der Vorgabe, daß immer Zellen fester Höhe – wie Bild 6.16 verdeutlicht - aneinander zu fügen sind, entsteht die charakteristische, in Bild 6.16 gut erkennbare Zellreihenstruktur, die geometrisch nur wenige Randbedingungen einhalten muß, wodurch eine hoher Automatisierungsgrad möglich wird. Die stark reduzierte Entwurfsaufgabe führt zu schnellen Ergebnissen. Das betrifft sowohl die Zeit für die Ersterstellung als auch für gegebenenfalls erforderliche Korrekturen.

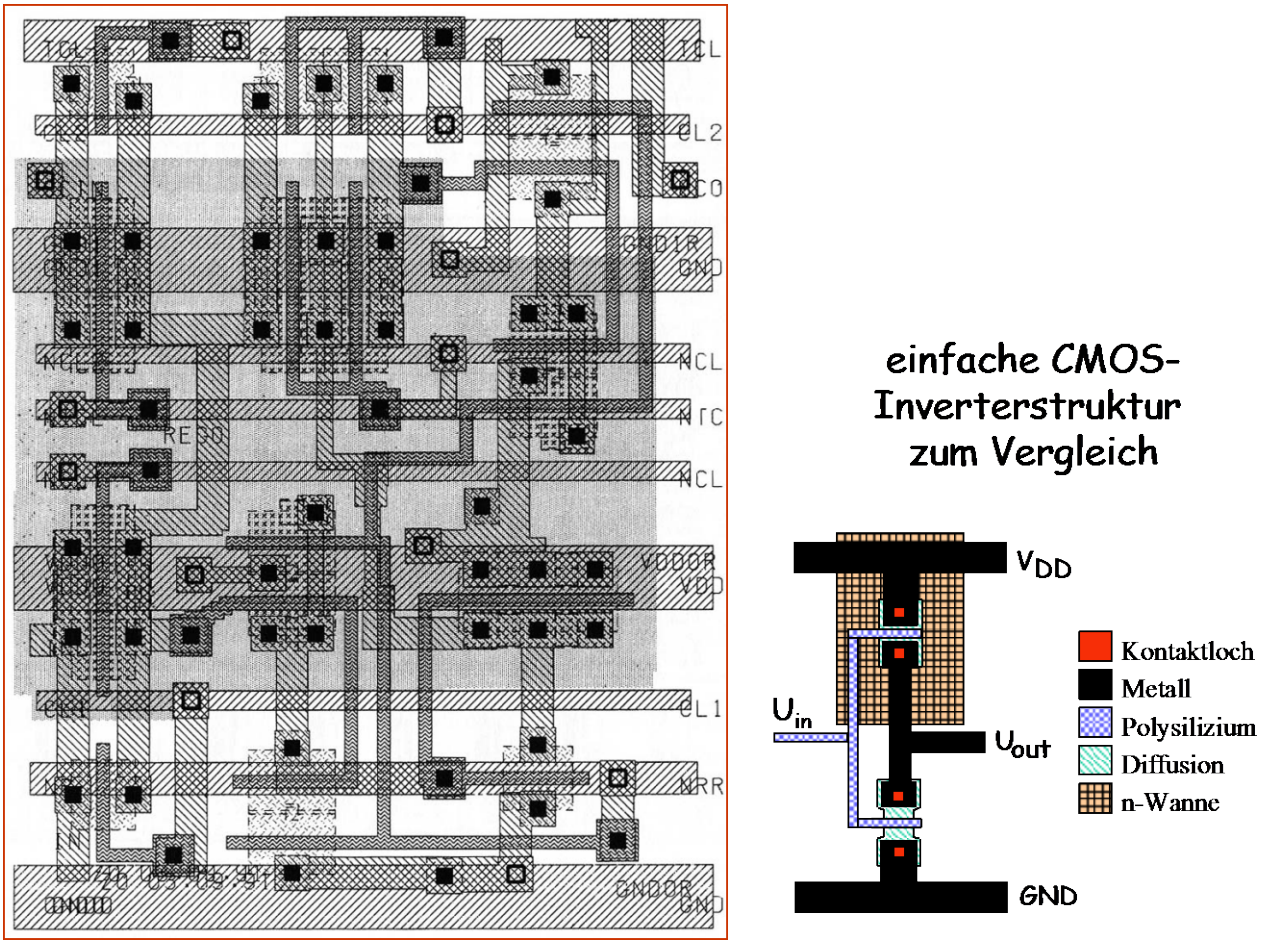


Bild 6.17: Einzelheiten einer Standardzellenstruktur im Vergleich mit einem CMOS–Inverter

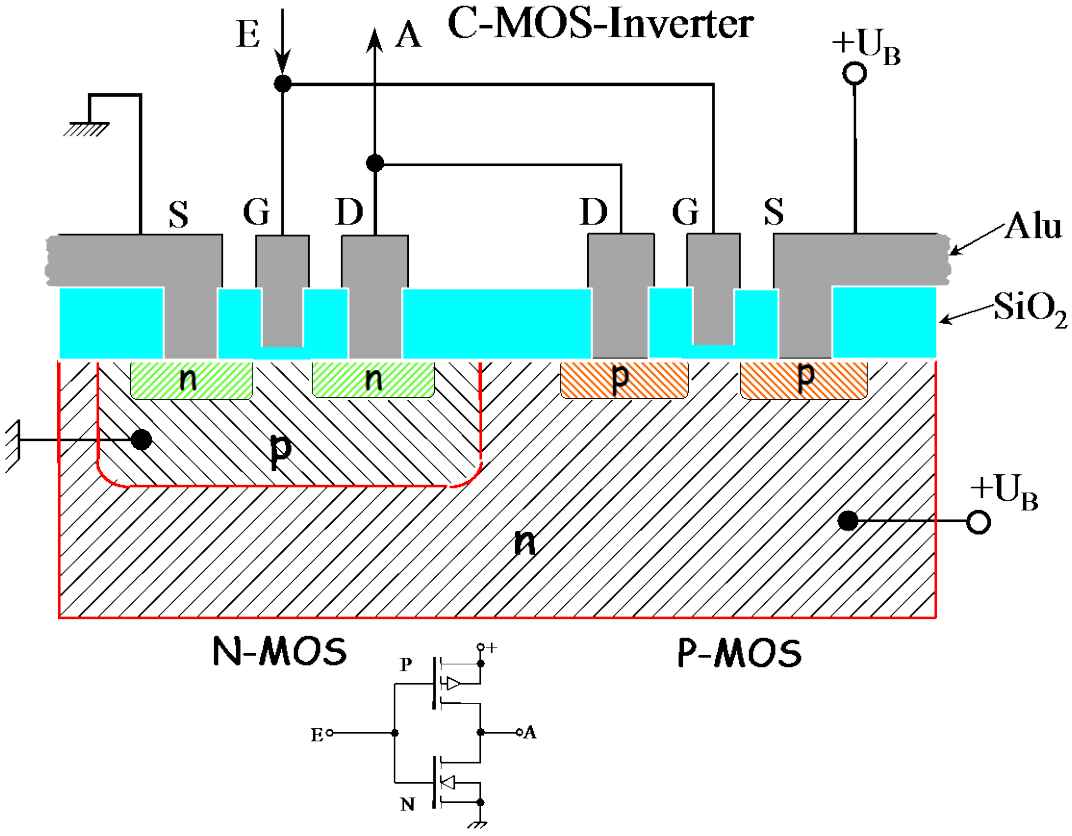


Bild 6.18: Querschnitt durch die Siliziumstruktur eines CMOS–Inverters

- geringes Fehlerrisiko durch verifizierte Zellen

  Geht man davon aus, daß sowohl die Zellen in der Bibliothek, als auch die Regeln, die zwischen den Zellen einzuhalten sind, verifiziert sind und ein korrektes Schaltungsverhalten garantieren, ist das Fehlerrisiko im Vergleich zum vollkundenspezifischen Entwurf erheblich reduziert. Was als Restrisiko verbleibt, ist die der Sicherstellung der korrekten Schaltungsfunktion unter Einbeziehung der Laufzeit- und Verzögerungseffekte durch die erzeugten Verbindungsstrukturen.

- wenig Optimierungsmöglichkeiten

  Da die Zellen auf Gatterebene festliegen, fehlt eine funktionale Strukturierung. Dies hat zur Folge, daß eine Optimierung der Zellanordnung, mit dem Ziel einer möglichst günstigen Verdrahtung, schwierig ist. EDA-Werkzeuge sind gewöhnlich nicht in der Lage, eine wirklich gute Anordnung zu finden. Auch bei manuellem Eingriff kann aufgrund des starren Anordnungsschemas immer noch eine ungünstige Zellanordnung verbleiben.

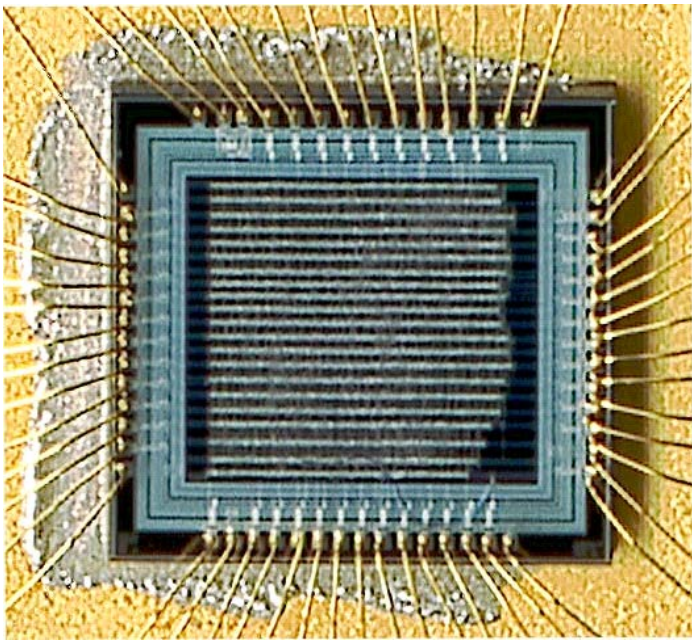
- Verschwendung von Verdrahtungsfläche

  Wenn die Anordnung der Zellen in den Reihen nicht geschickt gelöst werden kann, wirkt sich das nachteilig auf den Verdrahtungsaufwand aus. Bei einfachen Werkzeugen für Standardzellentwurf ist ein Verdrahtungsflächenanteil von 60-70 % der gesamten Chipfläche nicht ungewöhnlich.

- Der Chiprand (Padzellen) muß separat behandelt werden

  Die Anforderungen an die elektrischen Eigenschaften der Peripheriezellen, die die Anschlüsse nach Außen bilden, und die daraus resultierende Flächenaufteilung sind mit dem Standardzellentwurfsstil nicht verträglich. Daher ist in jedem Falle eine gesonderte Behandlung des Chiprandes und der Verdrahtung der Reihen mit den Padzellen erforderlich.

- einfacher, automatisierter Entwurf



Standardzellen sind im Hinblick auf einfache Automatisierbarkeit entstanden. Bei kommerziellen Entwurfssystemen für Standardzellen gibt es immer noch gravierende Nachteile, wie z.B.

– keine Garantie für vollständige Verdrahtung
– der Verdrahtungsflächenanteil kann unvertretbar hoch werden
– die Chipflächenausnutzung kann gering sein

Der Einsatz von Standardzellen ist immer dann empfehlenswert, wenn man einer kurzen Entwicklungszeit der Vorrang gegenüber einer hohen Technologieausnutzung geben kann.

Bild 6.19: FSK-Modem für die Powerline-Kommunikation in Form einer Standardzelle (realisiert im Rahmen von 'Europractice')

In der Praxis des Entwurfs digitaler mikroelektronischer Schaltungen wird der Standardzellentwurf heute am häufigsten verwendet, wenn sich die Stückzahlen im Bereich einiger Hunderttausend bis zu einer Million pro Jahr bewegen.

Bild 6.19 zeigt das Beispiel eines Standardzellenchips, der Mitte der 90-er Jahre am IIIT realisiert wurde. Er enthält die wesentlichen Funktionen des bereits mehrfach betrachteten digitalen FSK-Systems mit vierfachem Korrelator für inkohärenten FSK-Empfang – vgl. auch Bild 5.22. Aus heutiger Sicht ist die Komplexität von ca. 25.000 Transistoren eher gering.

Neben so genannten „Primitiven“ – wie z.B. Gattern, Flip-Flops - enthalten die heutigen Standardzellen-Bibliotheken der führenden Hersteller auch vielfältige „Makrozellen“. Diese sind praktisch ausschließlich VHDL–Modelle auf der Register-Transfer-Ebene. Man findet z.B. „generische[19]“ Beschreibungen von ALUs, Registerbänken, Steuerwerken oder Multiplexern bis hin zu kompletten Rechnerkernen (MP, MC und DSP). Mit heutigen Zellbibliotheken können vollständige, komplexe Systeme auf Silizium in kurzer Zeit erstellt werden. Dabei können auch analoge Schaltungsteile einbezogen werden, wenn sie als Bibliothekselemente verfügbar sind. Das ist z.B. der Fall für A/D- und D/A-Wandler oder Operationsverstärker. Voraussetzung ist natürlich, daß alle Analogfunktion mit demselben CMOS-Halbleiterprozeß hergestellt werden können, auf dem die Digitalfunktionen basieren. Eine Nutzung dieser Möglichkeit erfolgte 1997 im Rahmen eines Forschungsprojekts des IIIT mit der Fa. ABB. Hierbei wurde das anhand von Bild 5.28 bis Bild 5.30 beschriebene MFH-System in Form eines ’Mixed-Signal-ASIC’ realisiert.

Tabelle 6.8: Ausschnitt zur SFR–Definition des Mixed-Signal-ASICs

```
;Mixed-Signal-ASIC mit 8051 Prozessor und integrierter Peripherie
;Exemplarischer Auszug aus der INCLUDE-DATEI
;Zugriff auf Akkumulator-Werte für die Symbole 0...3 (Read Only)
Symbol0_L       XDATA   0FF80H
Symbol0_H       XDATA   0FF81H
Symbol1_L       XDATA   0FF82H
Symbol1_H       XDATA   0FF83H
Symbol2_L       XDATA   0FF84H
Symbol2_H       XDATA   0FF85H
Symbol3_L       XDATA   0FF86H
Symbol3_H       XDATA   0FF87H
;Betriebsarten-Auswahl für das Modem
Modem_Mode      XDATA   0FF02H
;Steuerregister der Filter und der automatischen Verstärkungsregelung AGC
OpAmp_Ctrl      XDATA   0FF08H
;Verstärkungseinstellung der Operationsverstärker
AGC_Gain_slow   XDATA   0FF07H
AGC_Gain_fast   XDATA   0FF0BH
;Signalformspeicher
WAV_F0B         XDATA   0F000H
WAV_F0E         XDATA   0F1F3H
WAV_F1B         XDATA   0F200H
WAV_F1E         XDATA   0F3F3H
WAV_F2B         XDATA   0F400H
WAV_F2E         XDATA   0F5F3H
WAV_F3B         XDATA   0F600H
WAV_F3E         XDATA   0F7F3H
WAV_QUAB        XDATA   0F800H
WAV_QUAE        XDATA   0F9F3H
```

Wie aus Bild 5.30 hervorgeht, umfassen die analogen Schaltungsfunktionen im wesentlichen Signalfilterung, Verstärkung sowie A/D- und D/A-Wandlung. Der mit „selektive DFT und Entscheidungslogik“ bezeichnete Block beinhaltet unter anderem eine MAC-Einheit mit 8 Akkumulatoren (8-facher digitaler Korrelator). Der Signalsynthesizer besteht im wesentlichen aus Adreßzählern und Abtastwertespeichern, wo die zu erzeugenden Signalformen abgelegt sind. Bedeutsam ist darüber hinaus die Integration des 8051-Mikrocontrollerkerns, der neben Aufgaben der Systemsteuerung und des Datentransfers (über seine serielle Schnittstelle) zudem die einfache Bedienung der gesamten Hardware über seinen Befehlssatz gestattet. Dabei kommt das Prinzip des ’Memory-Mapping’ zur Anwendung, wobei der Standard-Spezialfunktionsregistersatz des 8051 entsprechend

[19] hierbei wird mit Parameterübergabe erst bei der Instantiierung z.B. die Bitbreite festgelegt

erweitert wurde. Wegen der großen Zahl zusätzlich benötigter Register erfolgte eine Auslagerung in den externen Adreßbereich (XDATA), wobei natürlich die entsprechende Speicherhardware in Form eine RAM mitintegriert werden mußte.

Tabelle 6.8 zeigt exemplarisch einen Ausschnitt der so genannten 'Include'–Datei, die im 'Header' eines Assemblerprogramms aufgeführt werden muß, damit die über den 8051-Kern hinausgehende Hardware angesprochen werden kann.

6.6.3 Gate-Arrays

Über die Standardisierung von Zellen hinaus, ist auch eine Standardisierung der Chipfertigung selbst denkbar. Das wird bei Gate-Arrays praktiziert, wobei eine Vorfertigung der Siliziumscheiben, die alle Dotierungsschritte beinhaltet, realisiert ist. Dabei werden reguläre Anordnungen (Arrays) mit „Bauteilrümpfen" vorgefertigt, die dann mittels weniger Maskenschritte, die die Verdrahtung durchführen, für die gewünschte Schaltungsfunktion „personalisiert" werden. Die vorgefertigten Chips nennt man „Master". Beim Gate-Array besteht der Master aus Feldern mit Transistoren, die noch unverbunden sind, und aus Verdrahtungskanälen, die zwischen den Feldern angeordnet sind. Die typische Topologie ist dabei eine zeilenweise Anordnung, in der sich Transistorfelder und Verdrahtungskanäle abwechseln, oder eine matrixartige Anordnung von Transistorfeldern, die horizontal und vertikal von Verdrahtungskanälen getrennt werden.

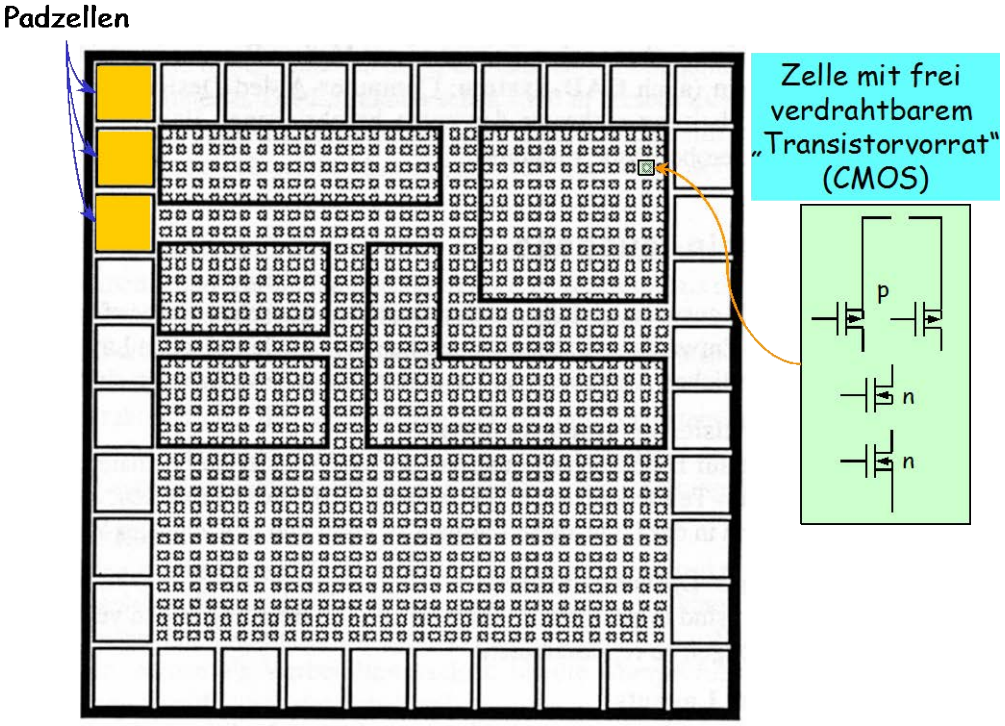


Bild 6.20 zeigt einen Gate-Array-Master, der die 'Sea-of-Gates'-Philosophie realisiert. Sie unterscheidet sich prinzipiell nur gering von der ursprünglichen Zeilenanordnung. Die Sea-of-Gates-Struktur ist durch moderne Halbleitertechnologie möglich geworden, die sehr hohe Integrationsdichten bereitstellt.

Bild 6.20: Gate-Array mit Sea-of-Gates-Struktur

Weil die Abmessungen der Transistoren drastisch reduziert werden konnten, hat sich eine Master-Topologie entwickelt, bei der der gesamte Chip mit einem dichten, zweidimensionalen Netz von Transistoren überzogen ist, zwischen denen relativ wenig Verdrahtungsraum bereitgehalten werden muß. Diese Topologie hat selbstverständlich Folgen für den Entwurf. Einerseits kann man nun Zellverdrahtungen mit recht hoher Komplexität zu erstellen, da nahezu beliebige zweidimensionale Verdrahtungsstrukturen möglich sind, insbesondere auch eine funktionale Orientierung. Bezüglich der Ausdehnung und der Form hat man lediglich durch das Transistorraster geringe Einschränkungen. Aufgrund der Regularität lassen sich Verdrahtungsstrukturen im Transistorraster verschieben, was insgesamt sehr flexible Layouts ermöglicht. Somit hat die Sea-of-Gates-Philosophie gegenüber

der weitaus starreren Zeilenstruktur die besseren Zukunftsaussichten, weil sie die Vorteile des flexiblen Entwurfs mit kurzen Fertigungszeiten verbindet.

Gate-Arrays aller Art werden generell nur für digitale Anwendungen eingesetzt. Der Entwurf geht in der Regel von einer Schaltungsbeschreibung auf Register–Transfer– oder Logikebene aus und besteht letztlich aus der Erstellung von Verdrahtungsmustern, die zunächst die Transistoren innerhalb der Felder zu logischen Primitiven (Inverter, Gatter, Flip-Flops) verbinden, und anschließend die Gesamtfunktion der Schaltung durch Verbindung der Primitive realisieren. Es stehen leistungsfähige EDA-Werkzeuge zur Verfügung, die auf vielfältige, vorentworfene Verdrahtungsmuster aus Bibliotheken zurückgreifen können. Eine gewisse Verwandtschaft zum Standardzellentwurf ist somit gegeben.

Neben den Einschränkungen, die der Standardzellentwurf aufweist, kommen bei Gate-Arrays weitere hinzu, die sich im wesentlichen aus der starren Architektur der Master ergeben. Als Vorteile sind dagegen hoher Automatisierungsgrad, schneller Fertigungszyklus und relativ niedrige „Einmalkosten“ (für die Maskenherstellung) anzuführen.

- geringe Ausnutzung der „theoretischen Chipkapazität“

  Die Vorfertigung der Master legt die theoretisch nutzbare Zahl von Transistoren auf dem Chip bereits fest. Diese Zahl kann jedoch bei Schaltungsrealisierungen praktisch nie erreicht werden. Die begrenzte Kapazität der Verdrahtungskanäle führt oft dazu, daß digitale Strukturen nicht in der wünschenswerten hohen Dichte angeordnet werden können. Man erhält relativ große Chips mit vergleichsweise geringer Komplexität.

- Mehrebenenverdrahtung verbessert die Chipausnutzung

  Moderne Technologien, die bis zu 6 Metallisierungsebenen für die Interzellverdrahtung zur Verfügung stellen, können den oben genannten Nachteil mindern, da sie eine Erhöhung der Verdrahtungskapazität bringen. Bei modernen Gate-Arrays kann so eine Chipausnutzung bis über 90 % erreicht werden, allerdings zu erhöhten Maskenkosten wegen der hohen Zahl der Metallebenen.

- automatisierter Entwurf ist nötig und möglich

  Da jegliche funktionale Orientierung beim Entwurfsprozeß fehlt, ist eine manuelle Durchführung praktisch ausgeschlossen. In der Tat ist ein entscheidendes Motiv für den Gate–Array–Einsatz durch die nahezu vollständig automatischen Abläufe gegeben.

- verkürzter Fertigungszyklus

  Da die Herstellung der Master universell, d.h. unabhängig von jedem Entwurf erfolgt, ist die Durchlaufzeit für eine Gate-Array-Produktion signifikant geringer als z.B. für Vollkundenschaltungen oder Standardzellen, die jeweils einen vollständigen Halbleiterherstellungszyklus durchlaufen müssen. Gate-Arrays erlauben einen ausgesprochen schnellen Durchlauf von der Fertigstellung des Entwurfs bis zur Verfügbarkeit der ASICs für den praktischen Einsatz (Time-To-Market-Vorteil).

Trotz der genannten Vorteile haftet auch den Gate-Arrays der Nachteil an, daß der Gang zum Halbleiterhersteller erforderlich ist, um zum einsetzbaren Chip zu kommen. Obwohl Kosten und Durchlaufzeiten im Vergleich zu Vollkundenschaltungen oder Standardzellen erheblich geringer sind, führen auch hier Fehler in der Regel zu Redesigns, die sich zeitraubend und kostenintensiv auswirken. Eine elegante Umgehung all dieser Probleme bieten heute FPGAs, deren Komplexität durchaus

mit Gate-Arrays vergleichbar ist. Der entscheidende Vorteil der FPGAs ist aber, daß sie am Arbeitsplatz des Entwurfs-Ingenieurs – wobei lediglich ein Standard–PC erforderlich ist – ohne Mitwirkung des Halbleiterherstellers mit der gewünschten Schaltungsfunktion versehen werden können. Zudem läßt sich dieser Vorgang bei den bevorzugten FPGA–Technologien beliebig oft wiederholen, so daß im Falle von Fehlern oder Verbesserungen so gut wie keine Zusatzkosten anfallen. Nicht zuletzt dieser Gesichtspunkt hat in jüngster Zeit zu enormem Zuwachs FPGA–basierter Designs geführt. Aufgrund der raschen Weiterentwicklung der technischen Anforderungen in allen Bereichen entpuppt sich eine „festgenagelte“ Schaltung – selbst wenn sie programmierbare Bestandteile beinhaltet – nach kurzer Zeit als zu unflexibel oder zu langsam, um auf neue Herausforderungen zu reagieren. Bevor derart hohe Stückzahlen erreicht werden, die eine Vollkundenschaltung oder Standardzelle rechtfertigen würden, fordert der Kunde oft schon Erweiterungen, die ein Redesign erforderlich machen würden.

Mit FPGA–Lösungen kann auf solche Wünsche schnell und kostengünstig reagiert werden. Der folgende Abschnitt befaßt sich mit dieser zukunftsträchtigen Thematik, um die wohl kaum ein künftiger Ingenieur der Elektrotechnik und Informationstechnik herumkommen wird.

6.6.3.1 Vom PLD zum Field Programmable Gate-Array (FPGA)

Die heutigen FPGAs haben ihren Ursprung in recht einfachen programmierbaren Logikbausteinen, die unter der Bezeichnung PLD (**P**rogrammable **L**ogic **D**evice) seit über 25 Jahren auf dem Markt sind. Die ersten PLDs dienten dazu, die so genannte Glue[20]–Logic zu vereinfachen, die oft hohe Anzahl von Standard–TTL- oder CMOS–Schaltkreisen zwischen Prozessoren externer Peripherie, wie z.B. Speichern, erforderlich machte. Bei Speichern wird meist die Decodierung von Adressen oder Adreßbereichen benötigt. Dazu sind große UND-Gatter – etwa mit 16...32 Eingängen – nötig. Solchen Anforderungen wurde durch die ersten einfachen PLDs Rechnung getragen.

Im folgenden wird exemplarisch ein einfacher PLD–Baustein vom Typ GAL 16V8 betrachtet. Der Gesamtaufbau eines GAL 16V8 ist in Bild 6.21 dargestellt. Der Baustein gliedert sich in 8 Zellen, von denen jede eingangsseitig über 8 UND-Gatter der Breite 32 verfügt, deren Ausgänge auf ein 8-faches ODER–Gatter führen, das sich jeweils in den mit OMLC bezeichneten Blöcken befindet. OMLC steht für **O**utput **L**ogic **M**acro **C**ell; eine Blockschaltung ist in Bild 6.22 zu sehen. Neben dem 8-fachen ODER enthält die OMLC in der Hauptsache ein vorderflankengetriggertes D-Flip-Flop, dem noch ein EXOR–Gatter (XOR) zur Invertierung des D-Eingangs vorgeschaltet ist. Die Konfiguration erfolgt über Multiplexer, die der Übersicht halber in Bild 6.22 nicht eingezeichnet sind. Alle eingangsseitigen UND-Gatter können per Konfiguration mit bis zu 32 Eingangssignalen verbunden werden. So viele wären zum einen nur dann verfügbar, wenn auch alle Ausgänge des Bausteins als Eingänge konfiguriert wären und zum anderen macht ein Verbinden von mehr als 16 Signalen keinen Sinn, da dann Signale direkt und negiert UND-verknüpft würden, was immer zu dem trivialen Resultat „0“ führen würde.

Mit PLD–Strukturen nach Bild 6.22 lassen sich boolesche Ausdrücke in der disjunktiven Normalform (ODER–Verknüpfung von Mintermen) unmittelbar abbilden. Mit Hilfe der Speicherelemente in den OMLCs können dann recht komplexe Automaten aufgebaut werden. Mit den 8 verfügbaren Flip-Flops könnten immerhin bis zu 256 Zustände kodiert werden.

Für allgemeine Anwendungen (Aufbau beliebiger Digitalschaltungen) erweist sich aber die starre UND-Matrix als wenig nützlich. Es ist weitaus besser, statt dessen eine höhere Zahl kleinerer Einheiten z.B. in Form von Wertetabellen (**L**ook **U**p **T**ables ≡ LUT - vgl. Bild 6.25) vorzusehen.

[20] das englische Wort für Klebstoff

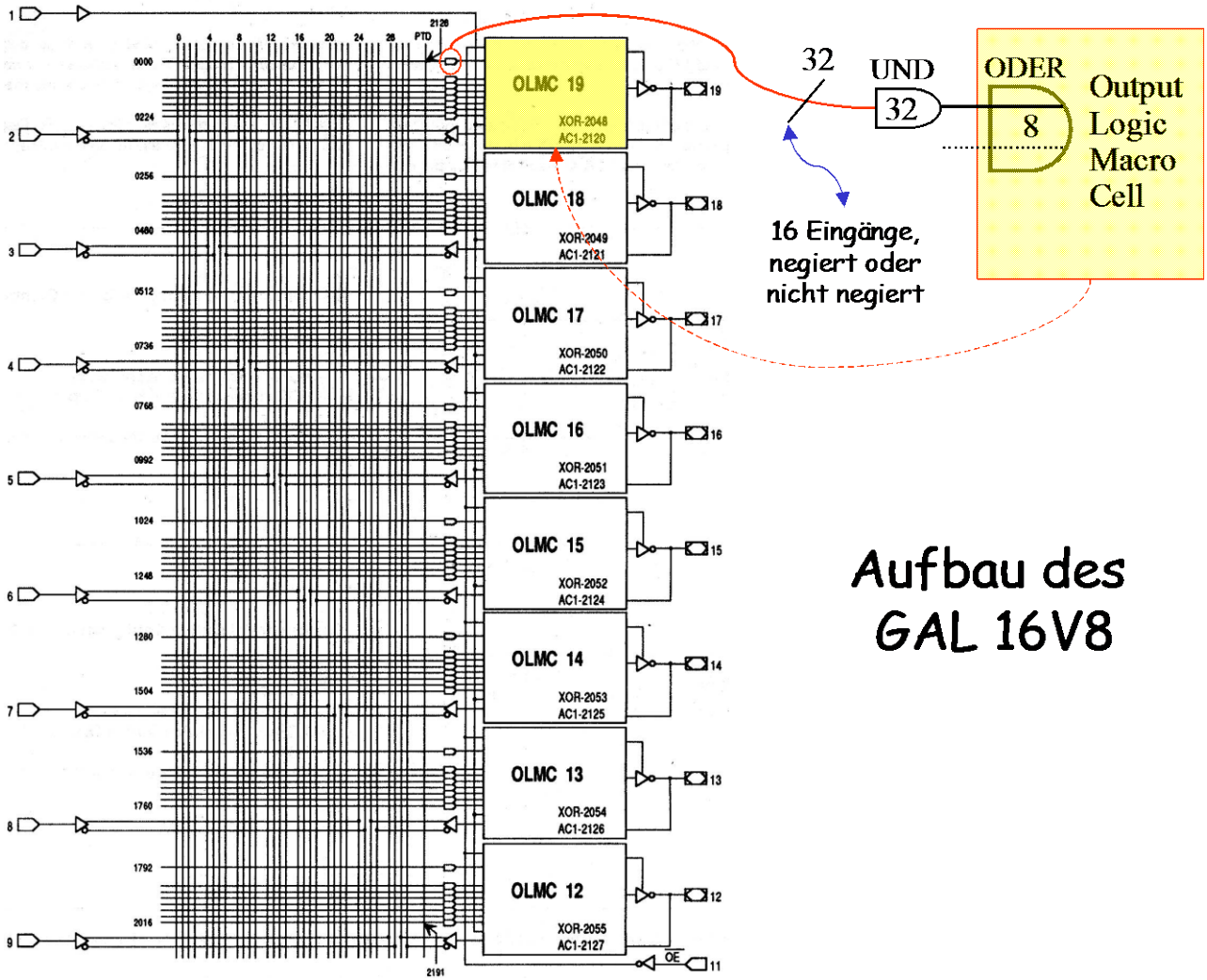


Bild 6.21: Aufbau des programmierbaren Logikbausteins GAL 16V8, bei dem die Konfigurierung mittels EEPROM–Zellen erfolgt

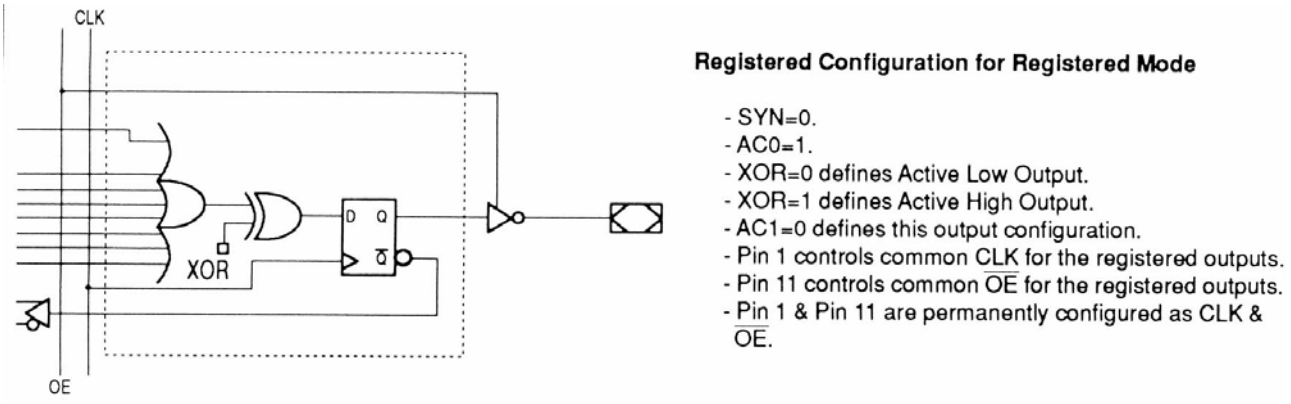


Bild 6.22: Konfiguration einer Output Logic Macro Cell als „registrierter“ Ausgang

Gemäß der Philosophie – mehr und kleinere Einheiten – sind aus den so genannten ’Simple’-PLDs, zu denen das GAL 16V8 zählt, die komplexen PLDs (CPLD) entstanden. CPLDs werden von zahlreichen Herstellern in großer Vielfalt angeboten. Bei genauerem Hinsehen lassen sich rasch gewisse bewährte Grundstrukturen ausmachen, die sich in mehr oder weniger modifizierter Ausgestaltung immer wieder finden. Es genügt daher für diese Einführung, exemplarisch und stellvertretend für viele andere einen Baustein aus der MAX 3000-Reihe der Fa. ALTERA zu betrachten – siehe Bild 6.23.

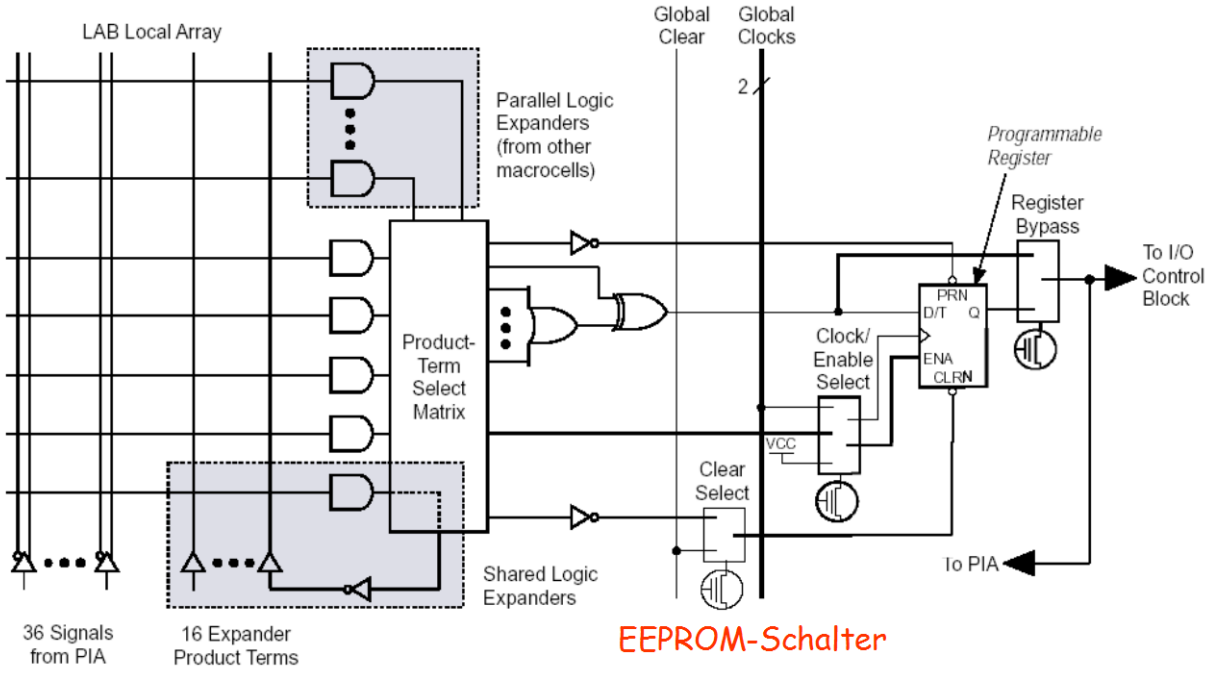


Bild 6.23: Typische Zellstruktur eines MAX 3000 CPLD–Bausteins

Die Konfiguration erfolgt ebenso wie beim GAL 16V8 mittels EEPROM-Zellen, allerdings mit dem wesentlichen Unterschied, daß kein Programmiergerät benötigt wird, denn der Baustein verfügt über eine JTAG-Schnittstelle, die eine ’In System’-Programmierung über ein einfaches Interface zum Standard-Druckerport eines PC ermöglicht. Auf diese Weise können im übrigen auch praktisch alle modernen FPGAs programmiert werden. Schaltung und Aufbau werden weiter unten vorgestellt.

Eingangsseitig erkennt man die bewährte UND-Matrix, aus der mit dem Block ’Produkt Term Select Matrix’ die gewünschte boolesche Funktion wiederum in disjunktiver Normalform erstellt werden kann. Ausgangsseitig hat man wieder ein Speicherelement, jedoch mit mehr Funktionen als beim einfachen PLD, die mittels mehrerer Multiplexer eingestellt werden können.

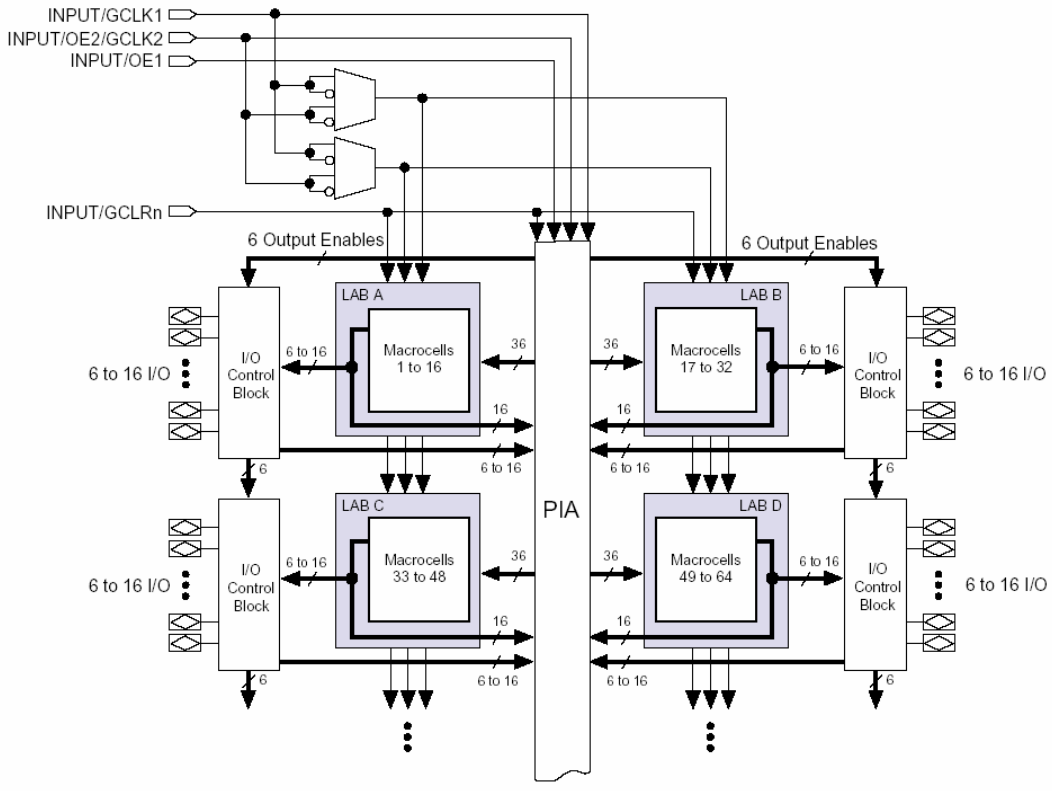


Bild 6.24: Gesamtaufbau eines MAX 3000 CPLDs

Der wesentliche Unterschied zum einfachen PLD wird in Bild 6.24 deutlich, wo der Gesamtaufbau des CPLDs zu sehen ist. Man sieht, daß hier jeweils 16 Zellen nach Bild 6.23 zu einem „**L**ogic **A**rray **B**lock“ (LAB) zusammengefaßt sind, von denen der hier dargestellte Baustein 4 Stück enthält. Man hat somit 64 Flip-Flops zur Verfügung, so daß sich schon sehr komplexe Automaten mit vielen Zuständen implementieren lassen. Neben den „I/O Control Blocks“, die jedem LAB zugeordnet sind, und in denen im wesentlichen festgelegt wird, ob ein Ausgang, ein Eingang oder eine Tristate-Funktion gewünscht ist, befindet sich im Zentrum eine komplexe Verdrahtungsmatrix, von deren Eigenschaften die erzielbare Geschwindigkeit der fertigen Schaltung wesentlich bestimmt wird. Hier kommt es darauf an, umständliches Routen zu vermeiden, d.h. die zahlreichen Signale, die hier zusammenkommen sind auf möglichst kurzen Wegen zu verschalten. In der Regel hängt die Leistungsfähigkeit einer implementierten Schaltung weit mehr von dieser Verdrahtungseffizienz ab als von Feinheiten des inneren Aufbaus der LABs oder deren Anzahl.
Der Schritt vom CPLD zum FPGA ist nun nicht mehr sehr groß, wie Bild 6.25 verdeutlicht.

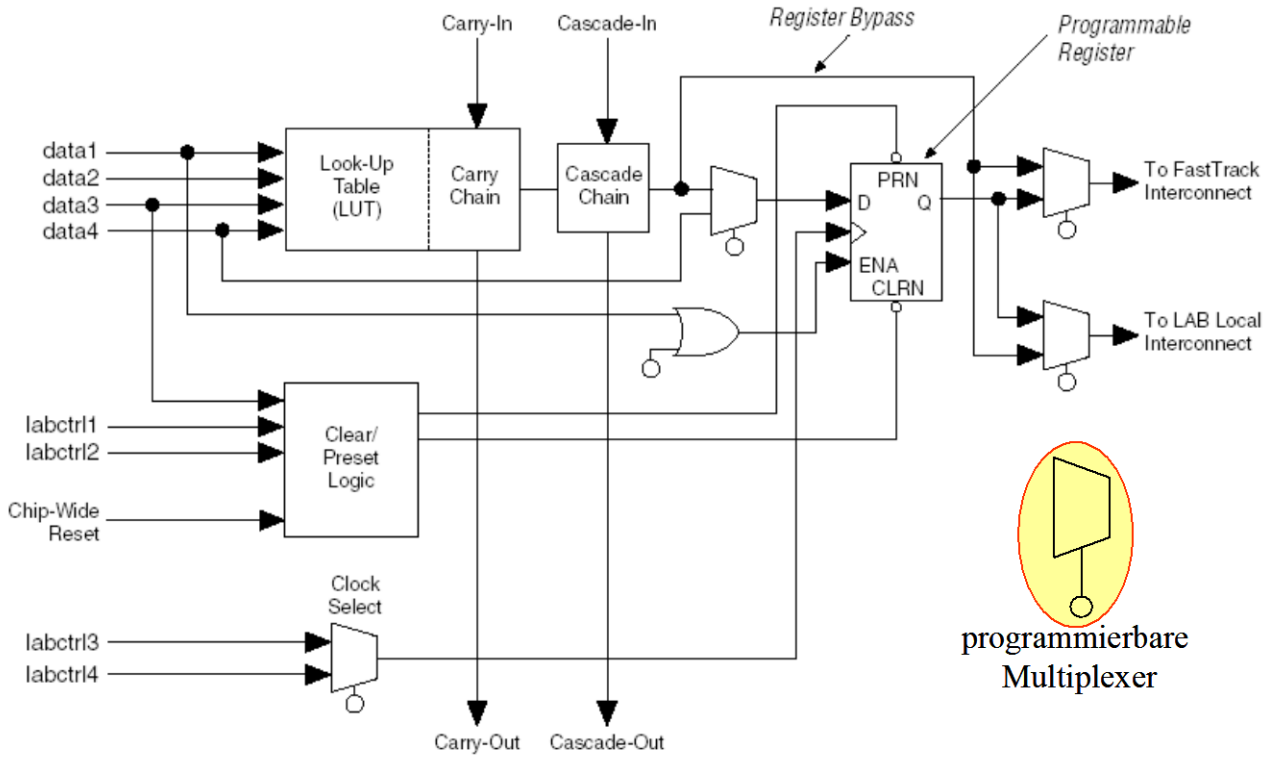


Bild 6.25: Logikelement (LE) eines FPGAs aus der ACEX-Familie (ALTERA)

Auffällig ist hier, daß die breitgefächerte UND-Funktion am Eingang nicht mehr vorhanden ist. An ihre Stelle ist eine flexible und viel allgemeiner einsetzbare Wertetabelle getreten. Der LUT-Block ermöglicht die Implementierung einer beliebigen booleschen Verknüpfung von 4 Eingangsvariablen. Am Ausgang eines Logikelements findet man wieder das vorderflankengetriggerte D-Flip-Flop mit erweiterten Eigenschaften (Enable ENA, Preset PRN und Clear CLRN). Mit Hilfe zahlreicher programmierbarer Multiplexer läßt sich das Element sehr flexibel konfigurieren. Neben übergeordneten Takt- und Resetsignalen lassen sich die Eigenschaften des LE mit „labctrl“-Signalen, die aus anderen Blöcken stammen, beeinflussen. Die Verkettung von LEs zu komplexen Funktionen wird durch Carry- und Cascade-Chains unterstützt.
In einer übergeordneten Hierarchiestufe werden aus den Logikelementen **L**ogic **A**rray **B**locks (LAB) gebildet – siehe Bild 6.26. Man sieht, daß hier 8 LE zusammengefaßt sind. Des weiteren sind in Bild 6.26 die Verdrahtungsmöglichkeiten angedeutet. Dabei werden zwei Hierarchieebenen unterschieden: LAB-Interconnect, für den „Nahbereich“, und Row- bzw. Column-Interconnect für „Fernverbindungen“.

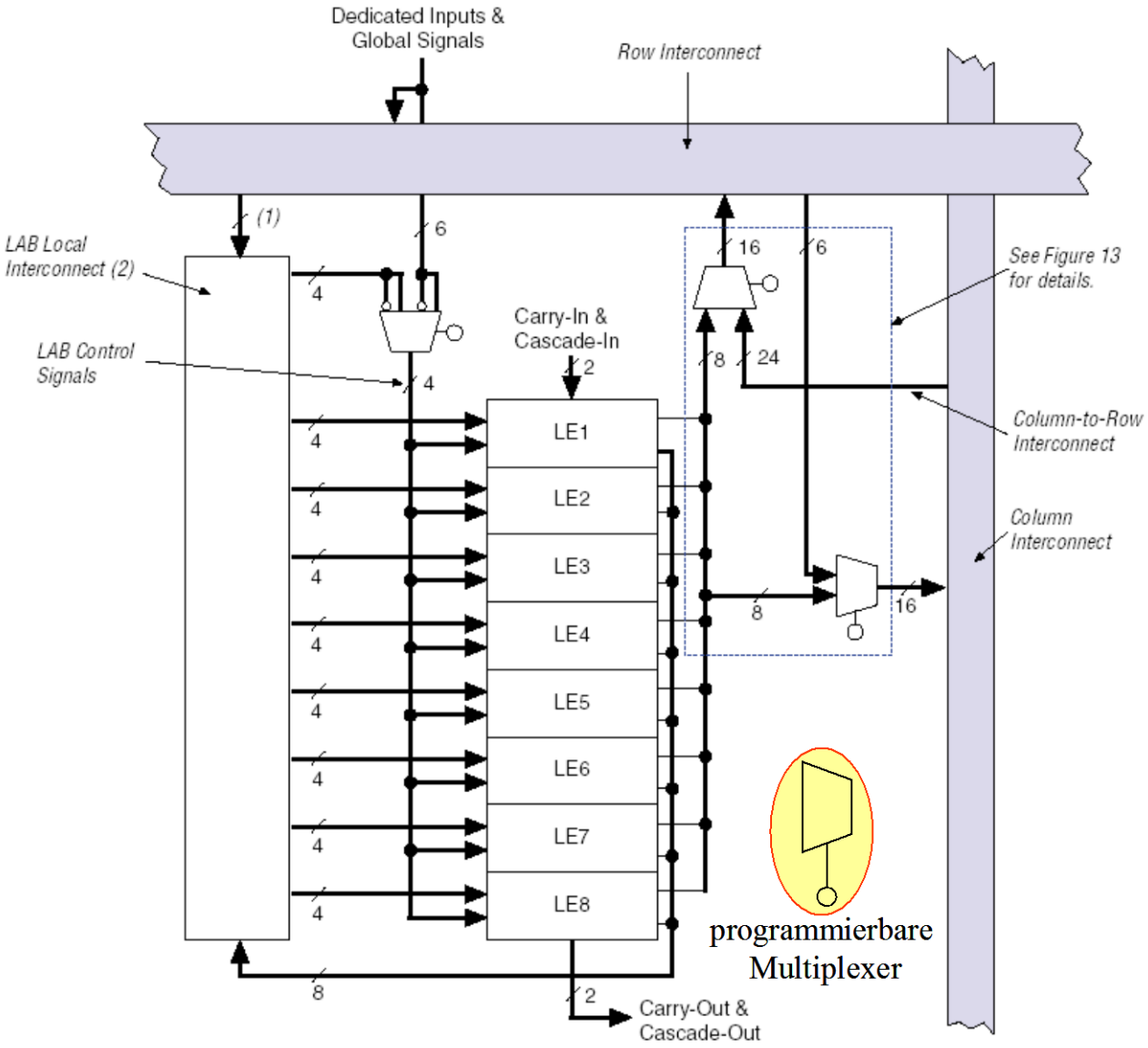


Bild 6.26: Aufbau eines LAB aus 8 Logikelementen bei ACEX FPGAs

Diesen Fernverbindungen kommt – wie schon beim CPLD angedeutet - eine hohe Bedeutung für die Leistungsfähigkeit und erzielbare Taktfrequenz der gesamten implementierten Schaltung zu. Das wird noch klarer beim Betrachten von Bild 6.27, wo der Gesamtaufbau eines ACEX-FPGAs dargestellt ist. Die matrixartige Verdrahtungsstruktur, die die Funktionseinheiten einrahmt, ist sehr augenfällig. Rechts und links im Bild ist die nächsthöhere Hierarchiestufe erkennbar, in der jeweils 8 LABs nach Bild 6.26 zu einem **L**ogic **A**rray (LA) kombiniert sind. Im gezeigten Beispiel stehen 8 LAs mit insgesamt 64 LABs, d.h. mit 256 Logikelementen (LE) zur Verfügung. Im Zentrum finden sich **E**mbedded **A**rray **B**locks (EAB), deren innere Struktur einen wesentlichen Beitrag dazu leistet, daß man komplette Prozessoren auf dem FPGA unterbringen kann. Hauptbestandteil eines EAB sind – wie Bild 6.28 zeigt – Speicherelemente in Form von RAM-Zellen, die sehr flexibel – besonders was die Wortbreite angeht – konfiguriert werden können. Diese Speicher wirken sich enorm ressourcensparend aus, da z.B. die Datenspeicher und Arbeitsregister von zu implementierenden Prozessoren sehr schnell die relativ wenigen Flip-Flops in den Logikelementen aufbrauchen würden. In den EABs kann man darüber hinaus mit ’verteilter Arithmetik’ auch schnelle Multiplizierer unterbringen.

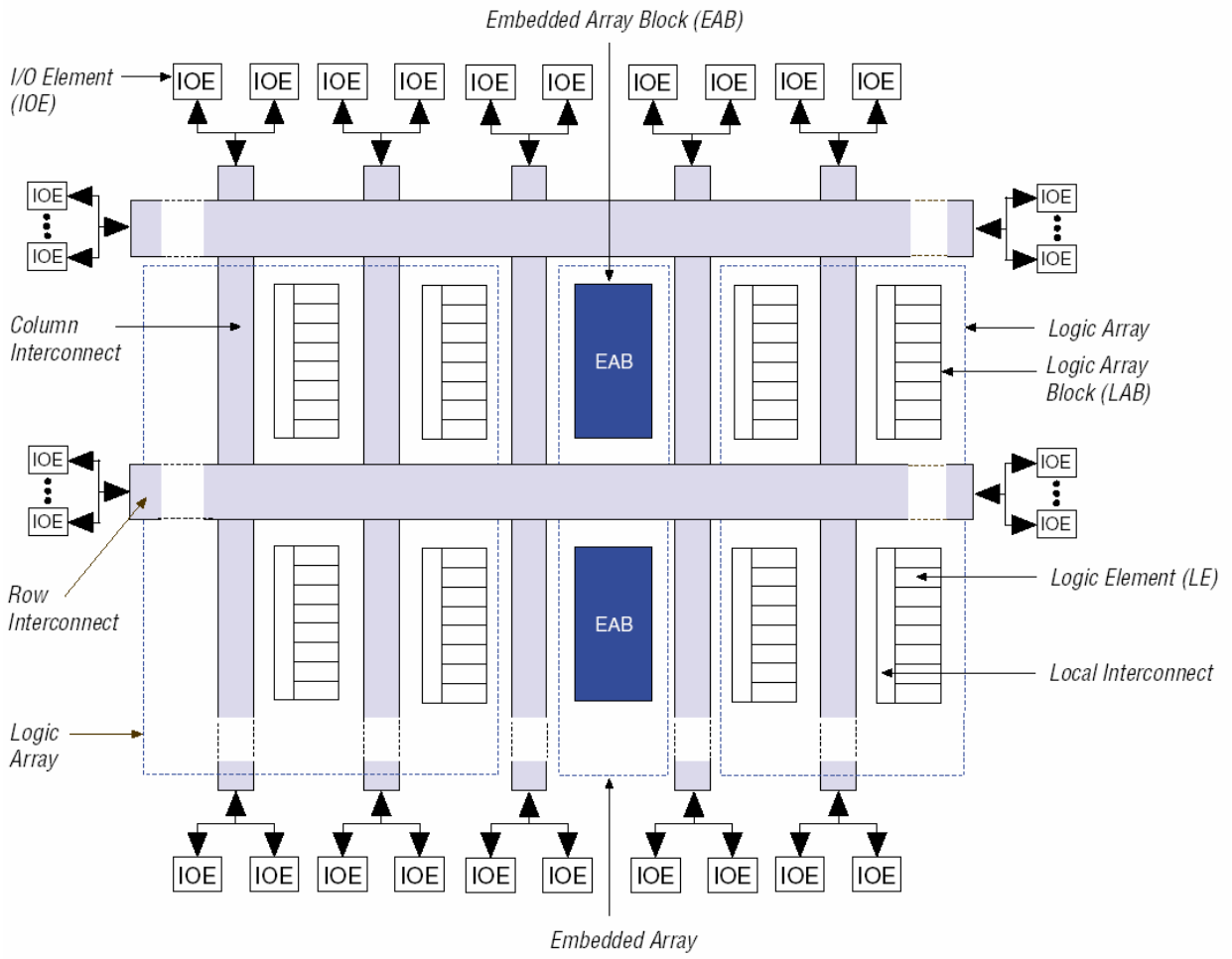


Bild 6.27:Gesamtarchitektur eines ACEX–FPGAs

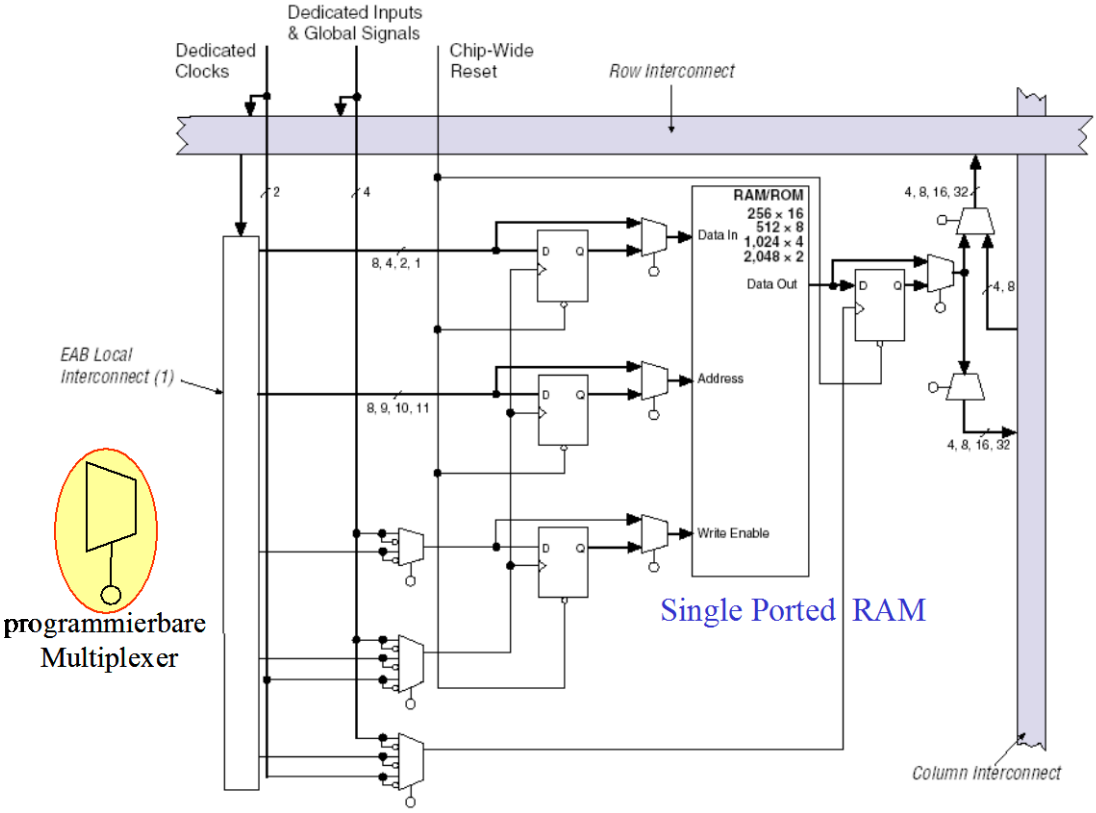


Bild 6.28: Embedded Array Block als ’single-ported RAM’ konfiguriert

Dazu gilt es, Algorithmen zu entwickeln, die das bereits vorgestellte speicherbasierte Multiplizierprinzip nutzen, wobei für lange Zahlen viele 'Untereinheiten' kaskadiert werden müssen. Aufgrund der hohen Zugriffsgeschwindigkeit der RAM-Zellen ist ein verteilt implementierter Multiplizierer trotz Kaskadierung immer noch sehr schnell. Für eine detailliertere Beschreibung der Möglichkeiten, verteilte Arithmetik zu realisieren und weitere Eigenschaften der FPGA-Architekturen vorteilhaft zu nutzen, wird auf wird auf umfangreiche Dokumentation verwiesen (ALTERA-WEB-Pages). Den Schluß dieses Abschnittes bildet eine ein kurzer Überblick über typische Eigenschaften der aktuellen Cyclone-III-Familie des Herstellers Altera.

Type: Cyclone III

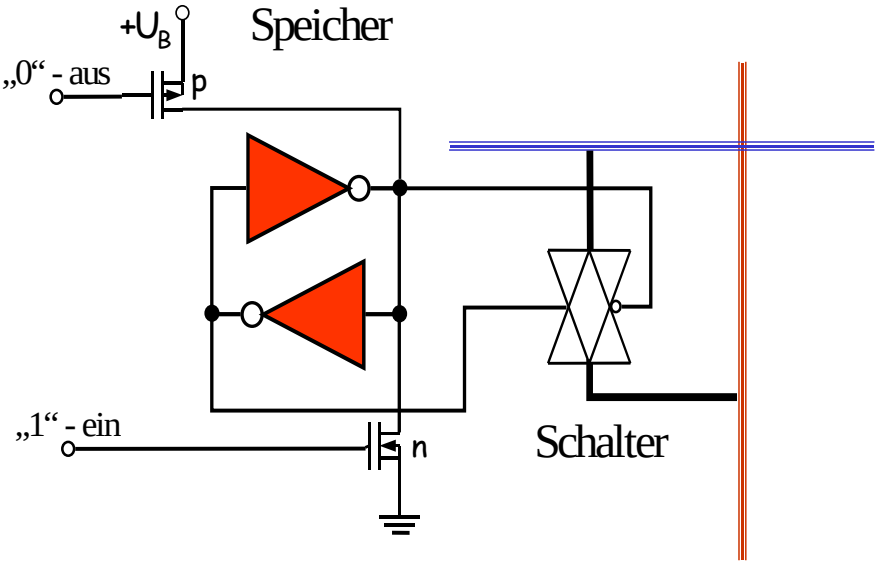
Feature	EP3C5	EP3C10	EP3C16	EP3C25	EP3C40	EP3C55	EP3C80	EP3C120
Logic Elements	5,136	10,320	15,408	24,624	39,600	55,856	81,264	119,088
Memory (Kbits)	414	414	504	594	1,134	2,340	2,745	3,888
Multipliers	23	23	56	66	126	156	244	288
PLLs	2	2	4	4	4	4	4	4
Global Clock Networks	10	10	20	20	20	20	20	20

Tabelle 6.9: Übersicht über die Eigenschaften Cyclone-III-FPGAs

Man erkennt in Tabelle 6.9 die abgestufte Komplexität, die sich z.B. in der Zahl der 'Logic Elements' widerspiegelt. Die Zahlen der Logikelemente (LE) und der Memory-Bits hingegen klare Anhaltspunkte für den Anwender. Des weiteren ist die Zahl der eingebauten Multiplizierer für komplexe Aufgaben der DSV von großer Bedeutung.

In den Forschungsarbeiten des IIIT wird derzeit bevorzugt der Typ EP3C55 eingesetzt. Seine Komplexität reicht mit mehr als 55.000 LEs, ca. 2,3 Mbit an Speicher und 156 Multiplizierern aus, um schnelle und aufwendige digitale Filter zu realisieren, d.h. die Rechenleistung von zahlreichen Standard-DSPs läßt sich problemlos implementieren. Des weiteren können Modulation und Demodulation für schnelle digitale Kommunikationssysteme für Datenraten >>10 Mbit/s realisiert werden.

6.6.3.2 FPGA und CPLD-Programmierung



Von den vielfältigen Möglichkeiten, PLDs, CPLDs und FPGAs zu konfigurieren, wird hier exemplarisch die mit einem bidirektionalen Schalter gekoppelte statische RAM-Zelle betrachtet.

In der Entwicklungsphase eines neuen Produkts hat diese Variante den Vorteil, daß die Zahl der Programmierzyklen unbegrenzt ist, und daß der Programmiervorgang auch bei komplexen Bausteinen sehr schnell abläuft.

Bild 6.29: Prinzip der FPGA-Konfiguration mittels statischer RAM-Zelle als Speicher und Transfergate als bidirektionalem Schalter.

Beim fertigen Produkt muß durch einen Festwertspeicher in Form eines Konfigurations-EEPROMs (meist mit 1-bit-Ausgang) dafür gesorgt werden, daß die Funktion beim Einschalten der Stromversorgung geladen wird. Das geht weitaus schneller als über den PC-Druckerport, d.h. in Bruchteilen einer Sekunde ist ein System betriebsbereit.

Die wesentlichen Bestandteile der Schaltung in Bild 6.29 sind ein CMOS-Transfergate und eine statische RAM-Zelle. Ein kurzes Anlegen eines „1“-Pegels an den n-Kanal-MOS-FET führt zum Einschalten des Schalters und dieser Zustand bleibt durch die Doppelinverterstruktur gespeichert, solange, bis ein „0“-Pegel an den p-MOS-FET angelegt wird, der den Speicher zum Kippen in den anderen Zustand bringt. Im Ruhezustand sind beide Transistoren (n- und p-Kanal) ausgeschaltet, so daß das Speicherelement aus den beiden Invertern seine Information behält bis eine Stromabschaltung erfolgt. Da es in der Praxis wichtig ist, daß alle Schalter kurz nach Anlegen der Betriebsspannung den gleichen Zustand – nämlich AUS – einnehmen, ist ein globaler Reset-Impuls an alle p-Kanal-Transistoren anzulegen, bevor die Konfiguration erfolgt. Dazu muß dann für jeden Schalter, der eingeschaltet sein soll, ein „1“-Impuls an den zugehörigen n-Kanal-Transistor gelegt werden. Die Übertragung dieser Information, die während des Placement– und Routing–Prozesses im PC erzeugt wurde, erfolgt über den Druckerport mit Hilfe einer einfachen, in Bild 6.30 dargestellten Interfaceschaltung, die im wesentlichen aus einem CMOS-Treiberbaustein (74HC244) besteht. Während die 100Ω-Widerstände für ein rasches Abklingen von leitungsbedingten Überschwingern an den Signalflanken sorgen, dienen die 2,2kΩ-Widerstände zur Anpassung der verschiedenen Betriebsspannungen (5V am PC und 3,3 V am FPGA). Bei dem ISP-Stecker handelt es sich um einen einfachen Flachkabelverbinder, der auf ein 10-adriges Flachkabel aufgepreßt wird. Geometrie und Pin-Numerierung sind gemäß Bild 6.32 genormt, während in Bild 6.31 die Pin-Funktionen kurz beschrieben sind. Hier wird die Verknüpfung mit der schon kurz angesprochenen Scan-Path- oder Boundary-Scan-Testmethodik klar – dort wurde auch bereits die Abkürzung JTAG eingeführt. Wie aus Bild 6.32 ersichtlich ist, können die einzelnen Pins je nach Anwendung (Programmierung oder Schaltungstest in der Halbleiterherstellung) verschiedene Aufgaben haben. Die Norm gibt lediglich einen Rahmen vor, der neben der Steckergeometrie auch die Stromversorgungspins (GND: Pins 2 und 10, VCC: Pin 4), die Taktversorgung (Pin 1), sowie Eingangspin (9) und Ausgangspin (3) allgemein definiert.

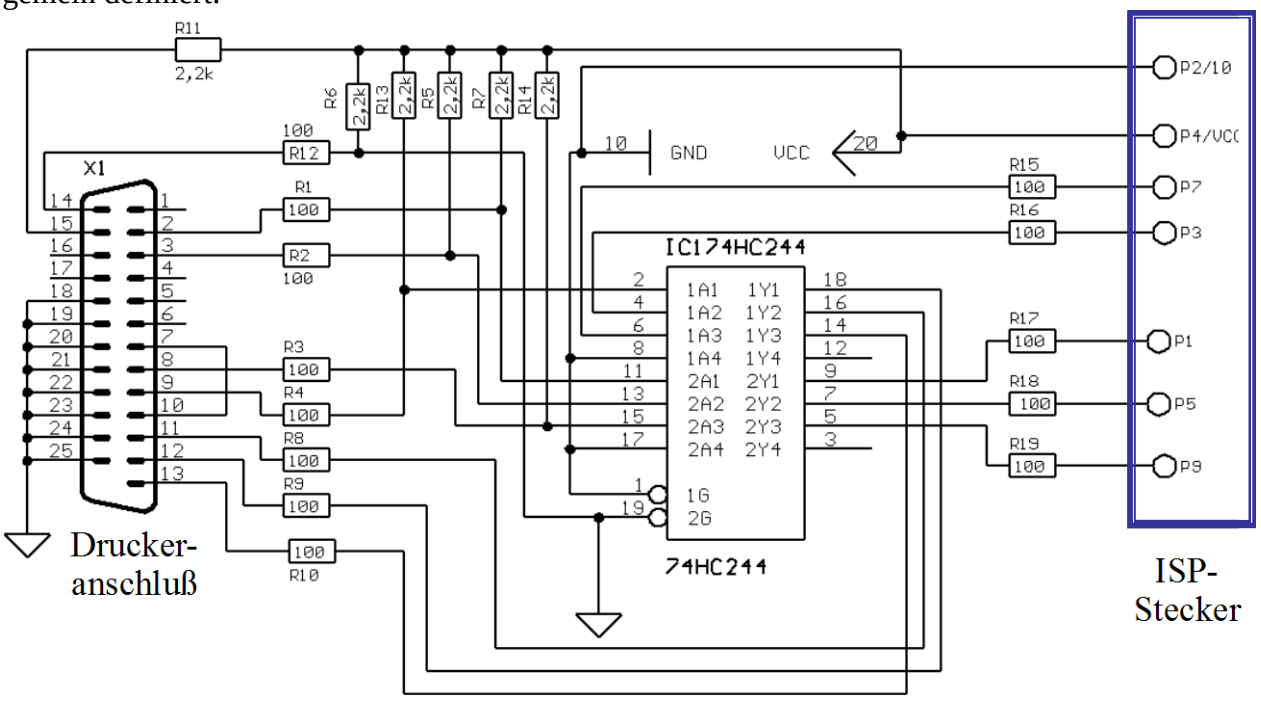


Bild 6.30: Interface zwischen PC-Druckerport und „In-System-Programmierstecker“

Bild 6.31 stellt die Programmieranwendung (PS-Mode), die uns hier interessiert, der Testanwendung (JTAG-Mode) gegenüber. Man sieht, daß zum Programmieren neben dem Takt und dem einzuschreibenden Datenstrom (DATA0) noch ein Steuerbit (nCONFIG) und zwei Überwachungsbits (nSTATUS und CONF_DONE) vorgesehen sind.

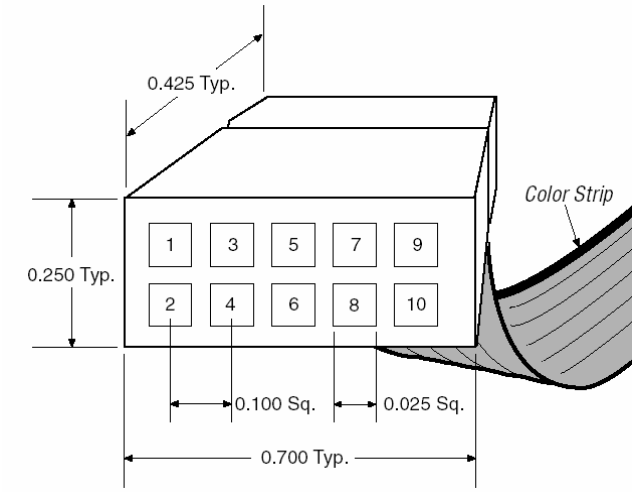


Bild 6.31: Aufbau und Belegung des 10-poligen ISP-Steckers

Pin	PS Mode		JTAG Mode	
	Signal Name	Description	Signal Name	Description
1	DCLK	Clock signal	TCK	Clock signal
2	GND	Signal ground	GND	Signal ground
3	CONF_DONE	Configuration control	TDO	Data from device
4	VCC	Power supply	VCC	Power supply
5	nCONFIG	Configuration control	TMS	JTAG state machine control
6	–	No connect	–	No connect
7	nSTATUS	Configuration status	–	No connect
8	–	No connect	–	No connect
9	DATA0	Data to device	TDI	Data to device
10	GND	Signal ground	GND	Signal ground

Bild 6.32: Pin-Funktionen am ISP-Stecker

Im JTAG-Testbetrieb müssen Testvektoren über Pin 9 (TDI) in die interne Flip-Flop-Kette eingeschrieben und die Testergebnisse über Pin 3 (TDO) ausgelesen werden.

Beim Arbeiten mit FPGAs muß man sich nicht um Details der Signalübertragung für die Programmierung kümmern. In der PC-basierten FPGA–Entwicklungsumgebung QUARTUS II der Fa. Altera, die hier kurz exemplarisch betrachtet wird, steht die Menüfunktion „Programmieren“ zur Verfügung, die für die Übertragung der Konfigurationsinformation über Druckerport und ISP-Stecker ins Ziel-FPGA sorgt. Der Vorgang ist in Bild 6.33 anschaulich zusammengefaßt.

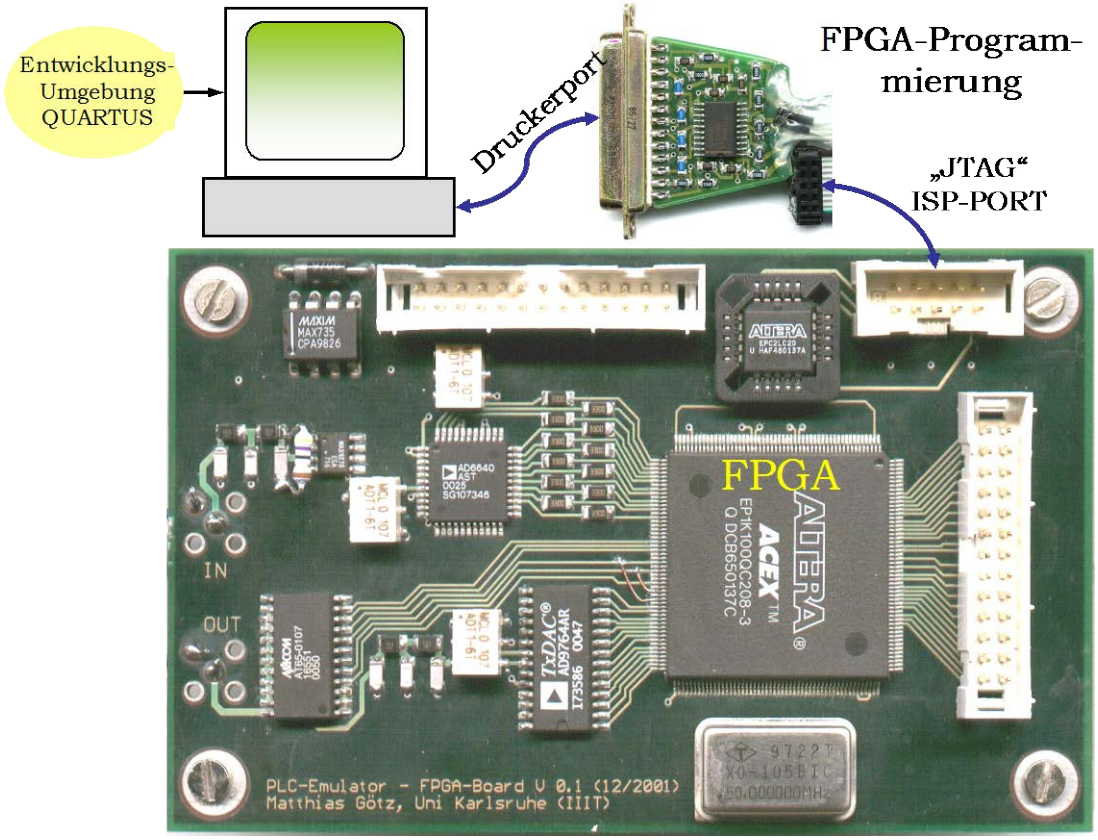


Bild 6.33: Illustration des Programmiervorganges bei FPGAs

Im oberen Bildteil ist neben dem PC die Platine der Schaltung nach Bild 6.31 zu sehen, die das Bindeglied zwischen PC und Anwenderhardware bildet. Mit dem FPGA wird hier ein komplexer Powerline-Kanalemulator realisiert. Links neben dem ISP–Stecker erkennt man einen kleinen quadratischen Baustein in einem Sockel. Hierbei handelt es sich um das Konfigurations-EEPROM, aus dem das FPGA automatisch beim Anlegen der Stromversorgung programmiert wird, wenn die Entwicklung abgeschlossen und die PC-Verbindung entfernt ist.

Bild 6.34 gibt einen Einblick in die FPGA–Entwurfsumgebung QUARTUS II. Die Software belegt ca. 1GB Festplattenplatz und läuft auf PCs unter Windows oder Linux, kann aber auch unter UNIX auf Workstations eingesetzt werden. Um auch sehr große FPGAs, z.B. mit mehr als 10 Millionen 'Typical Gates' bearbeiten zu können, sollte ein PC über mindestens 4GB Arbeitsspeicher verfügen. Für genügend Rechenleistung sollte ein Prozessor mit mindestens 3 GHz Taktfrequenz benutzt werden.

In QUARTUS II steht ein komfortabler VHDL–Editor zur Verfügung. Des weiteren gibt es umfangreiche Bibliotheken mit logischen Primitiven und auch Makros, von denen viele allerdings extra gekauft, bzw. lizenziert werden müssen. Hiermit kann sowohl in VHDL als auch mit dem Grafikeditor – vgl. Bild 6.34 – gearbeitet werden. Hinter jedem der Blöcke steht ein VHDL–Modell, das an die zu lösende Aufgabe angepaßt werden kann. Wenn die Design-Eingabe komplett ist, kann die Synthese auf die Zielhardware hin erfolgen. Es entsteht eine Netzliste, auf deren Basis die Schaltung mit den Bauteileigenschaften simuliert werden kann. Nachdem Plazierung und Verdrahtung erfolgt sind, ist eine erneute Simulation, die den Einfluß der Verdrahtung berücksichtigt, erforderlich. Bestätigt diese Simulation die gewünschte Funktion der Schaltung, kann der Programmierer aufgerufen werden, der die Konfigurationsdaten wie oben beschrieben ins Ziel-FPGA bringt.

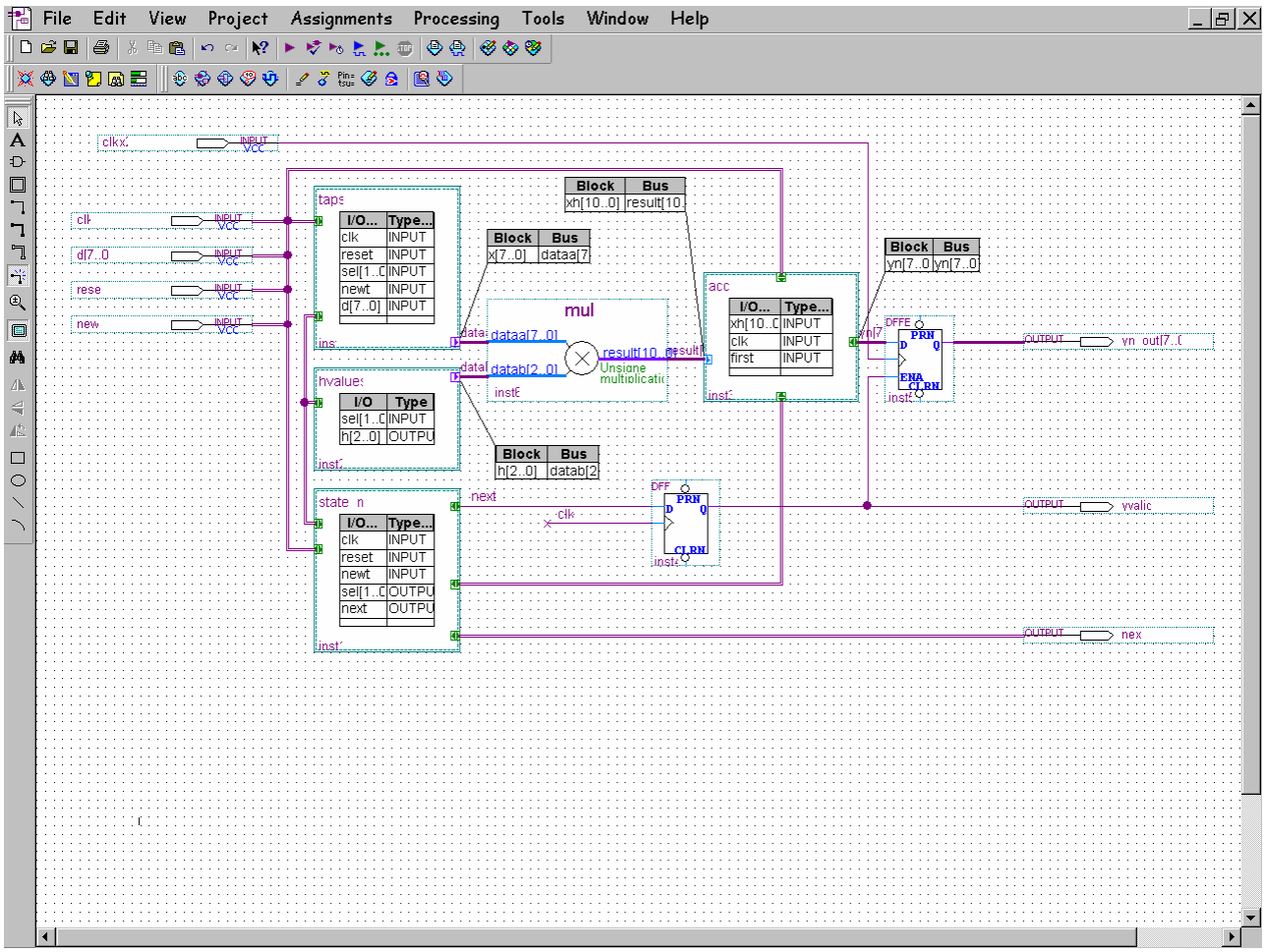


Bild 6.34: Einblick in die FPGA–Entwurfsumgebung QUARTUS II (Grafik Editor)