Universität Karlsruhe (TH)
Institut für Mikro– und Nanoelektronische Systeme

Praktikum

„Schaltungsdesign mit FPGA“

Karlsruhe, den SS 2013

Postanschrift: Institut für Mikro– und Nanoelektronische Systeme
Hertzstraße 16
D - 76128 Karlsruhe
Gebäude: Hertzstraße 16, Geb. 06.41

Tel.: +49 (0) 721 608 4 49 61
Sekr.: +49 (0) 721 608 4 49 61
Fax.: +49 (0) 721 75 79 25
Email: doris.duffner@kit.edu
WWW: http://www.ims.kit.edu

Inhaltsverzeichnis

1 Einleitung

In der digitalen Schaltungstechnik werden heute neben den bekannten Serien von Logikbausteinen niedriger bis mittlerer Komplexität bzw. Integrationsdichte in bipolaren und Feldeffekt- Technologien wie TTL-Logik-Familien, ECL-Familien und CMOS-Reihen immer mehr programmierbare Logikbausteine eingesetzt.

Diese programmierbaren Bausteine liegen in der Komplexität zwischen diesen Standard-Logik-Serien und den kundenspezifischen integrierten Logikschaltungen, wie sie z.B. in Mikrorechnersystemen eingesetzt werden.

Während das Arbeiten mit den Standard-Serien auf einem Logikentwurf mit bestimmten bekannten und festgelegten Funktionen basiert, hat man beim Einsatz von programmierbaren Logikbausteinen die Möglichkeit, die verschiedensten logischen Funktionen auf einem einzigen Baustein zu realisieren. Je nach Art und Aufbau des Logikbausteins kann man damit bis zu 100 Standardbausteine ersetzen.

Dies beinhaltet jedoch ein fundiertes Wissen über den logischen Aufbau dieser Bausteine und umfangreiche Hilfsmittel von der Eingabe der Schaltung über die Simulation bis zur Programmierung der Bausteine.

In diesem Praktikum soll der Umgang mit verschiedenen Typen von programmierbaren Logikbausteinen anhand von Beispielen erlernt und geübt werden. Hierzu wurde ein Entwicklungssystem der Firma ALTERA gewählt, da hier eine große Auswahlmöglichkeit unter verschiedenen Bausteinfamilien gegeben ist.

1.1 Versuchsdurchführung und Benotung

Das Praktikum wird im CIP–Pool der Westhochschule, Geb. 06.41 Raum 102, durchgeführt. Sie benötigen dafür Ihren Stud–Account und einen USB Stick für die lokale Datenspeicherung, da die Sicherheitsrichtlinien des RZ es leider unmöglich machen, Ihnen Speicherplatz auf dem Institutsserver zu überlassen. Die verfügbare Speichergröße des USB Sticks sollte mindestens 256 MB betragen. Das Praktikum beinhaltet drei unterschiedliche Projekte (Faltungscodierer, LED–Lauflicht und digitale Filter) mit entsprechenden Unterprojekten. Die Beschreibungen zu jedem Versuch mit einem Theorieteil finden Sie ab Kapitel 6 dieses Skriptes. Daraus werden sechs Teilnoten gebildet. Die siebte Teilnote ergibt sich aus dem Praktikumsbericht, der nach Ende des Praktikums zu erstellen ist. Halten Sie Ihre Praktikumsergebnisse deshalb in einem schriftlichen Bericht mit Screenshots und Ausarbeitung Ihrer Ergebnisse bzw. Schlussfolgerung fest und geben sie diesen nach Ende des Praktikums per Email beim Betreuer ab. Die Gesamtnote

ergibt sich dann aus dem geometrischen Mittel der Teilnoten.

2 Grundlagen

In diesem Kapitel sollen die Bezeichnungen bzw. Gruppen der Logikbausteine erläutert und auf den prinzipiellen schaltungstechnischen Aufbau eingegangen werden. Die programmierbaren Logikbausteine lassen sich nach Bild 2.1 grob wie folgt einteilen: Die Grundlage für die Entwicklung programmierbarer Logikbausteine ist das PROM,

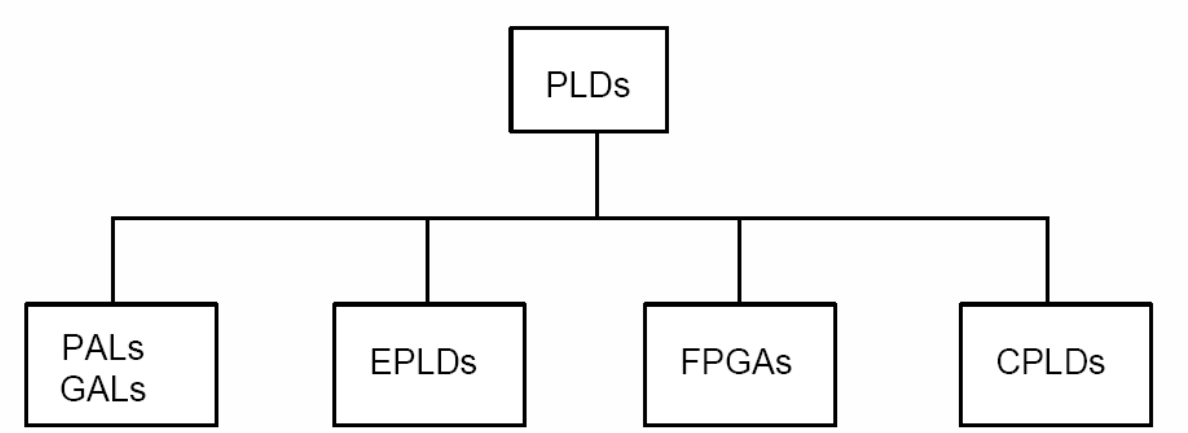

Bild 2.1: Grobe Übersicht der programmierbaren Logikbausteine.

ein vom Anwender einmalig programmierbarer Speicherbaustein. Ein Vergleich der Programmierschemata (Bild 2.2) soll die wesentlichen Unterschiede der Bausteinfamilien aufzeigen. Bei einem PROM stellen die Eingänge Adressen dar, deren Inhalt bei entsprechender Auswahl am Ausgang abgegriffen werden kann. Bei einem PAL–Baustein sind die Ausgangssignale fest zugeordnet, während die Aufteilung der Eingänge vom Anwender festgelegt werden kann. Den höchsten Freiheitsgrad für den Anwender bieten die FPGAs, denn hier kann sowohl das UND–Array auf der Eingangsseite wie auch das ODER–Array auf der Ausgangsseite programmiert werden.

Zur Gruppe der CPLDs (**C**omplex **P**rogrammable **L**ogic **D**evice) gehören die LCAs (**L**ogic **C**ell **A**rrays) der Firmen Xilinx und AMD und die EPLDs (**E**rasable **P**rogrammable **L**ogic **D**evice) und die FLEX–Bausteinserie von ALTERA. Diese werden in einem späteren Kapitel ausführlich beschrieben.

2.1 PAL's

Die erste Gruppe von programmierbaren Logikbausteinen auf dem Markt wurden mit der Bezeichnung PAL (**P**rogrammable **A**rray **L**ogic) versehen.

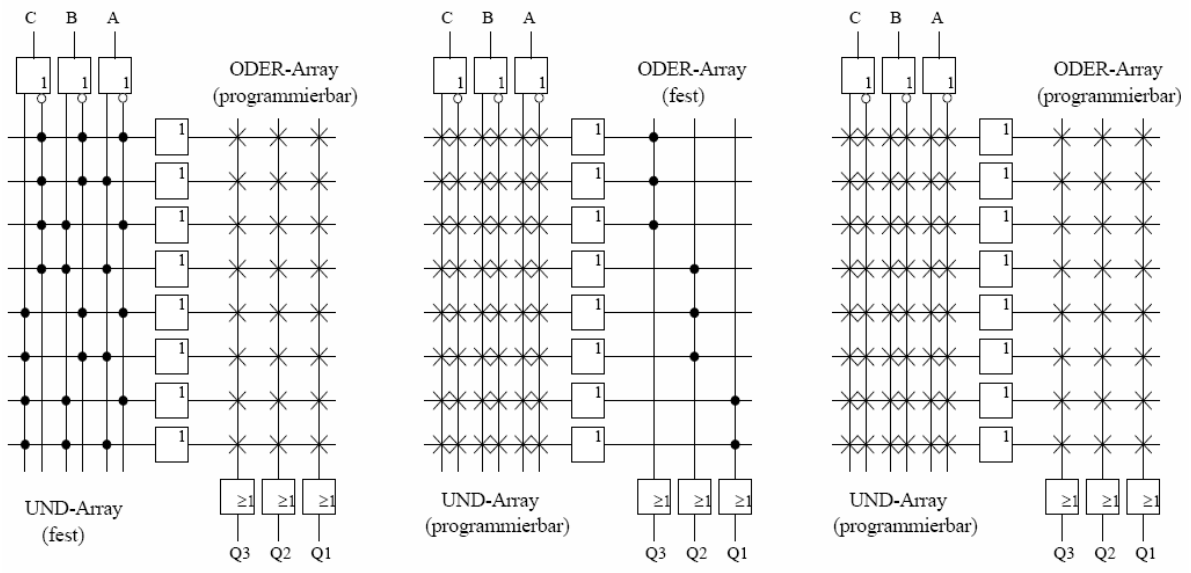

Bild 2.2: Schematische Struktur eines PROMs, PALs und FPGAs (v. l. n. r.).

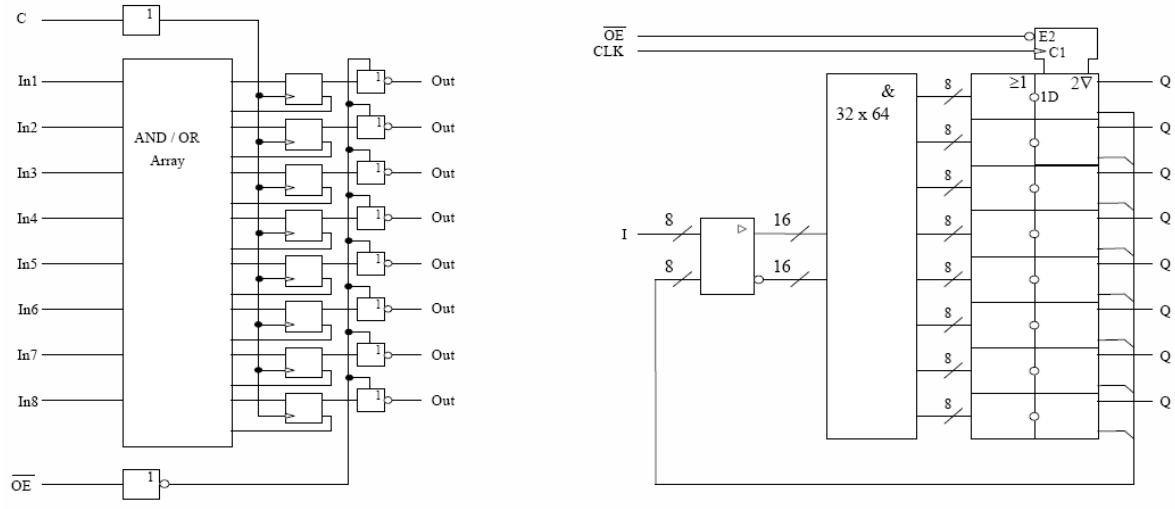

Bild 2.3: Blockschaltbild und logisches Symbol eines PAL 16R8.

PAL x nn y mm -tt T P	
x: Eingangsregister-Typ	ohne Bez.: kein Eingangsregister R: D-Flipflop T: transparentes Flipflop
nn: Anzahl der Eingänge des Eingangsfeldes	
y: Angaben zum Ausgang	R: mit Ausgaberegister L: Ausgang aktiv LOW X: Ausgang EXOR-verknüpft V: Variabler Ausgang, z.B. über Makrozelle
mm: Anzahl der Ausgänge des Bausteins	
-tt: Angabe über Geschwindigkeit des Bausteins	Laufzeit (in ns) oder Ziffer
T: Angabe über Temperaturbereich	z.B.: C: 0 ... 70°C M: -55 ... +125°C
P: Angabe über Gehäusetyp und -material	N: DIP, Plastik J: DIP, Keramik NW: DIP 600-mil, Plastik FN: Chip Carrier, Plastik FK: Chip Carrier, Keramik u.a.m.

Tabelle 2.1: Aufgeschlüsselter Zahlen- und Buchstabencode von PALs.

Ein PAL ist ein durch den Benutzer programmierbarer Baustein auf der Basis von schmelzbaren Mikro–Brücken (wie bei den PROMs). Die Eingänge können über programmierbare UND–Funktionen miteinander verknüpft werden deren Ausgänge über festverdrahtete ODER–Funktionen verknüpft sind. Die Bausteine sind mit einem Zahlen- und Buchstabencode bezeichnet, welchem genaue Aussagen über den internen Aufbau zu entnehmen sind (vgl. Tabelle 2.1). Ein Baustein mit der Bezeichnung PAL 16 R 8 -12 C N hat also ein Eingangsfeld mit 16 Eingängen ohne Register, 8 Ausgänge mit Register, 12 ns Gatterlaufzeit, ist für einen Temperaturbereich von 0 bis 70° C spezifiziert und ist in einem Standard Dual–In–Line Plastikgehäuse mit 0,3" Abstand der Pinreihen.

Neben der Bezeichnung PAL existieren für bestimmte Funktionen weitere Bezeichnungen, wie z.B. PAD = Programmierbarer Adreßdekoder, PSL = Programmierbare Sequenz Logik. Diese Bezeichnungen können jedoch von Herstellerfirma zu Herstellerfirma unterschiedlich sein.

Nach dieser Erklärung soll der prinzipielle Aufbau eines PAL anhand von logischen Symbolen und Schaltbildern näher betrachtet werden. Bild 2.3 zeigt das Logische Symbol eines PAL 16R8 Bausteins. Die oben beschriebenen 16 Eingänge des UND/ODER–Arrays setzen sich aus 8 Eingängen des Bausteins (In1–In8) und aus den Rückführungen der 8 Flipflops am Ausgang des UND/ODER–Arrays zusammen. In dem UND/ODER–

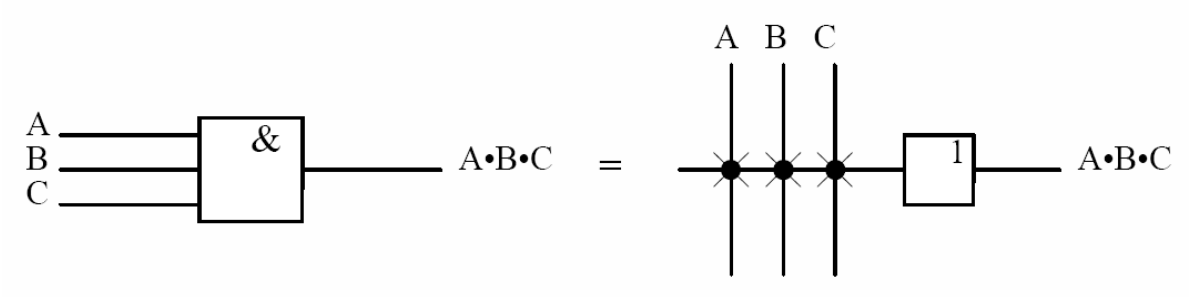

Bild 2.4: Beispiel einer UND–Verknüpfung.

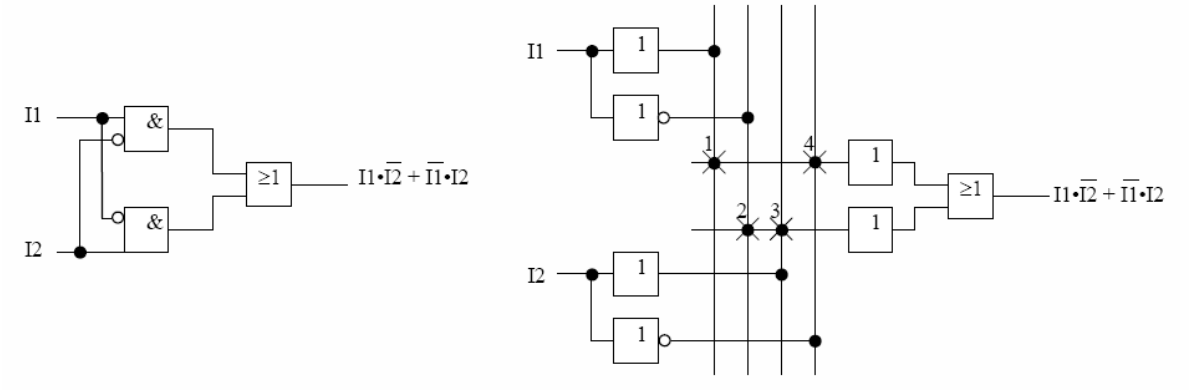

Bild 2.5: Logisches Schaltbild und Realisierung der Verknüpfung zweier Eingänge bei einem.

Array werden logische Verknüpfungen durch die Programmierung eines Knotens realisiert. Bei einem PAL–Baustein ist dieser Knoten zunächst ein intakter Knoten, d.h. es besteht eine elektrische Verbindung am Kreuzungspunkt der Leitungen, die beim Programmieren zerstört wird wenn keine elektrische Verbindung hergestellt werden soll. Anhand der Beispiele in Bild 2.4 und Bild 2.5 sollen einfache logische Verknüpfungen und deren Realisierung mit einem programmierbaren Logikbaustein gezeigt werden. Die durchkreuzten Punkte stellen dabei programmierbare Knoten dar. Von den in Bild 2.5 dargestellten 4 programmierbaren Knoten (1–4) bleiben je nach gewünschter Funktion 2 bestehen, während die anderen beiden zerstört werden müssen. Soll z.B. eine ODER–Verknüpfung realisiert werden, so dürfen nach dem Programmieren nur noch die Verbindungen der Knoten 1 und 3 bestehen bleiben.

Der prinzipielle Aufbau eines PAL–Bausteins mit Ausgangsregistern ist in Bild 2.6 dargestellt. Das Ergebnis der sogenannten Produktterme wird mit der ansteigenden Flanke des Taktsignals im Ausgangs–D–Flipflop gespeichert. Wenn der Ausgangstreiber freigegeben ist, wird die Information am Q–Ausgang des Registers invertiert am Ausgangspin des Bausteins anliegen. Gleichzeitig wird die Ausgangsinformation sowohl invertiert wie auch nichtinvertiert zurückgeführt und kann bei entsprechender Programmierung des Bausteins beim nächsten Taktschritt als Eingangsinformation verwendet

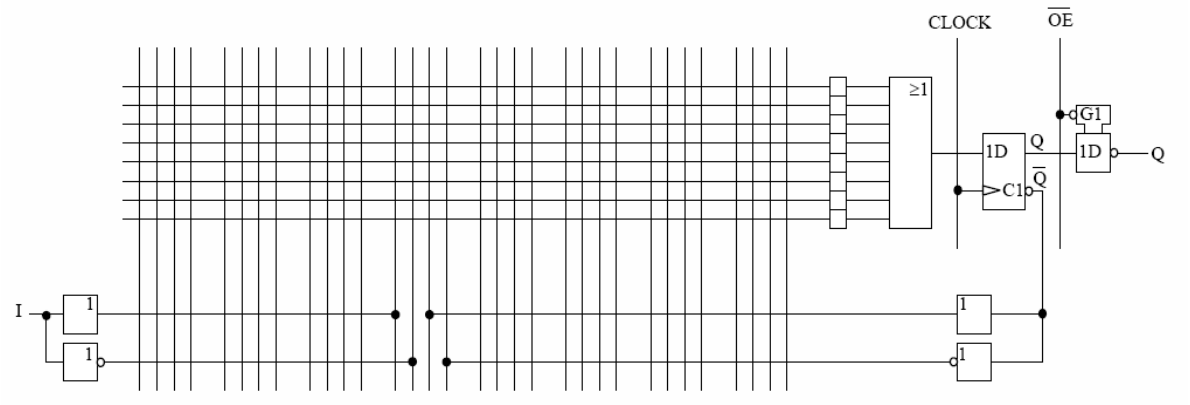

Bild 2.6: Darstellung der Struktur eines Eingangs und Ausgangs des PAL 16R8.

werden. Damit können unter anderem auch Funktionen wie Zählen und Schieben realisiert werden. Das Ausgangsregister in Bild 2.6 kann bei anderen Ausführungen von PLDs auch fehlen, bzw. durch eine sogenannte Makrozelle, welche im allgemeinen sowohl ein Eingangs- wie auch ein Ausgangsregister besitzt und an einen I/O–Pin des Bausteins führt.

2.2 GAL's

Eine Erweiterung der Gruppe der PALs hat die Bezeichnung GAL (**G**eneric **A**rray **L**ogic). Sie besitzen im wesentlichen die gleiche Struktur, haben jedoch gegenüber den PAL–Bausteinen ein- und ausgangsseitig eine erheblich erweiterte Matrix.

2.3 EPLD's

Die nächst komplexere Bausteinfamilien sind die EPLDs. Sie basieren im allgemeinen auf einer CMOS EPROM–Technologie, d.h. diese Bausteine können im Gegensatz zu den PALs mehrfach programmiert werden, da sie wie ein normales EPROM mit ultraviolettem Licht gelöscht werden können. In neueren Bausteinserien wird inzwischen die EEPROM–Technik eingesetzt, d.h. das Löschen erfolgt hierbei elektrisch. Damit entfällt die aufwendige Gehäusetechnik der EPROMs, die ja ein Quartzglasfenster besitzen müssen. Die EPLDs sind aus sogenannten Makrozellen aufgebaut, die durch eine Art Bus miteinander verbunden sind. Die Komlexität reicht von 8 bis hin zu mehreren Hundert Makrozellen, also von einem bis zu einigen Zehn PALs. Damit lassen sich schon recht umfangreiche Logikfunktionen auf einem einzigen Baustein realisieren. Die prinzipiellen Strukturen dieser Bausteine sollen anhand einiger Darstellungen näher erläutert werden.

Die EPLD–Bausteine besitzen im allgemeinen einige reine Eingänge und eine Reihe von Ein- und Ausgängen. Die speziellen Takteingänge führen direkt zu den Makrozellen.

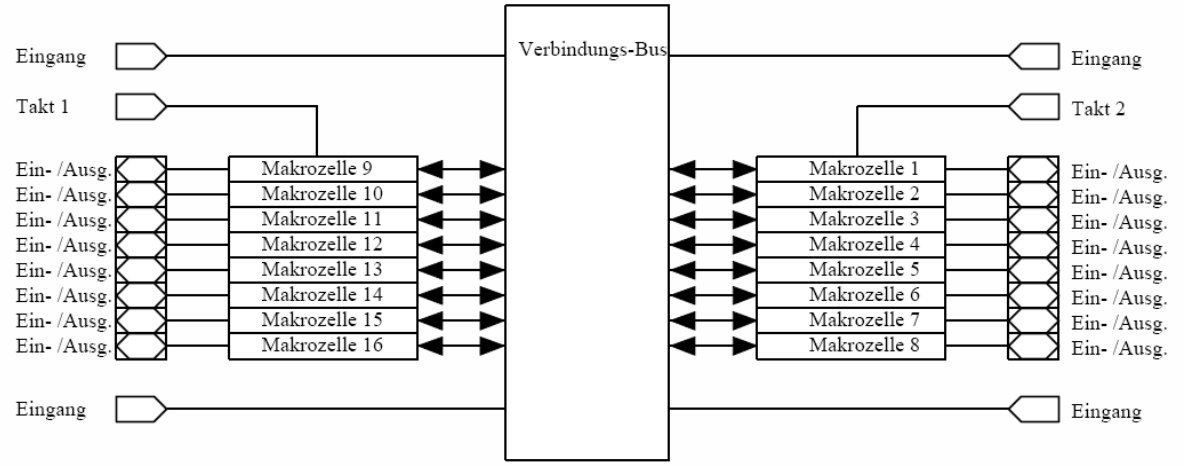

Bild 2.7: Aufbau eines sogenannten klassischen EPLDs.

Selbstverständlich kann auch an jeden anderen Eingang ein Taktsignal angelegt werden. Die Makrozellen besitzen alle ein frei programmierbares Flipflop, das vom Anwender entweder als D–, T–, SR– oder JK–Flipflop oder als Element für kombinatorische Logik eingesetzt werden kann. Die programmierbare UND/feste ODER–Logik kann zur Implementierung von Logikelementen mit bis zu 8 Produkttermen eingesetzt werden. Die programmierbare Ein–Ausgangs–Logik kann sowohl für Signale mit aktivem HIGH–Pegel wie auch mit aktivem LOW–Pegel verwendet werden. Den Aufbau einer Makrozelle zeigt Bild 2.8.

Durch die höhere Komlexität der EPLDs ist es notwendig, vor der endgültigen Auswahl des verwendeten Bausteins, sich einen Überblick zu verschaffen, welche zeitlichen Verzögerungen innerhalb des Bausteins zwischen Eingang und Ausgang auftreten können. Dazu soll das Übersichtsschema in Bild 2.9 herangezogen werden. Der Übergang von den klassischen programmierbaren Bausteinen zu den komplexen PLDs ist fließend. Die Frage, ab welcher Anzahl vom Makrozellen oder ab welcher Zahl von Produktterm en ein Baustein als komplexer Baustein zu bezeichnen ist kann in dieser Form nicht beantwortet werden. Die Struktur dieser EPLDs soll am Beispiel eines EPLDs aus der Serie MAX7000 erläutert werden (Bild 2.10).

Der Baustein MAX7064 besitzt 64 Makrozellen und hat eine programmierbare I/O–Architektur mit bis zu 68 Eingängen oder 64 Ausgängen. Die Auffächerung der Produktterme der Expander ist konfigurierbar und erlaubt bis zu 32 Produktterme in einer einzelnen Makrozelle. Der Baustein besitzt 1250 Gatteräquivalente und kann mit bis zu 125 MHz getaktet werden. Die Makrozellen dieser Bausteine werden zu sogenannten „Logical Array Blocks“ (LABs) zusammengefaßt. Die Makrozellen selbst besitzen ein programmierbares UND/festes ODER–Array und ein frei konfigurierbares Register, das Möglichkeiten wie einen unabhängigen, programmierbaren Takteingang, einen Taktfreigabeeingang und Lösch- und Setzfunktionen bietet. Die LABs und damit die

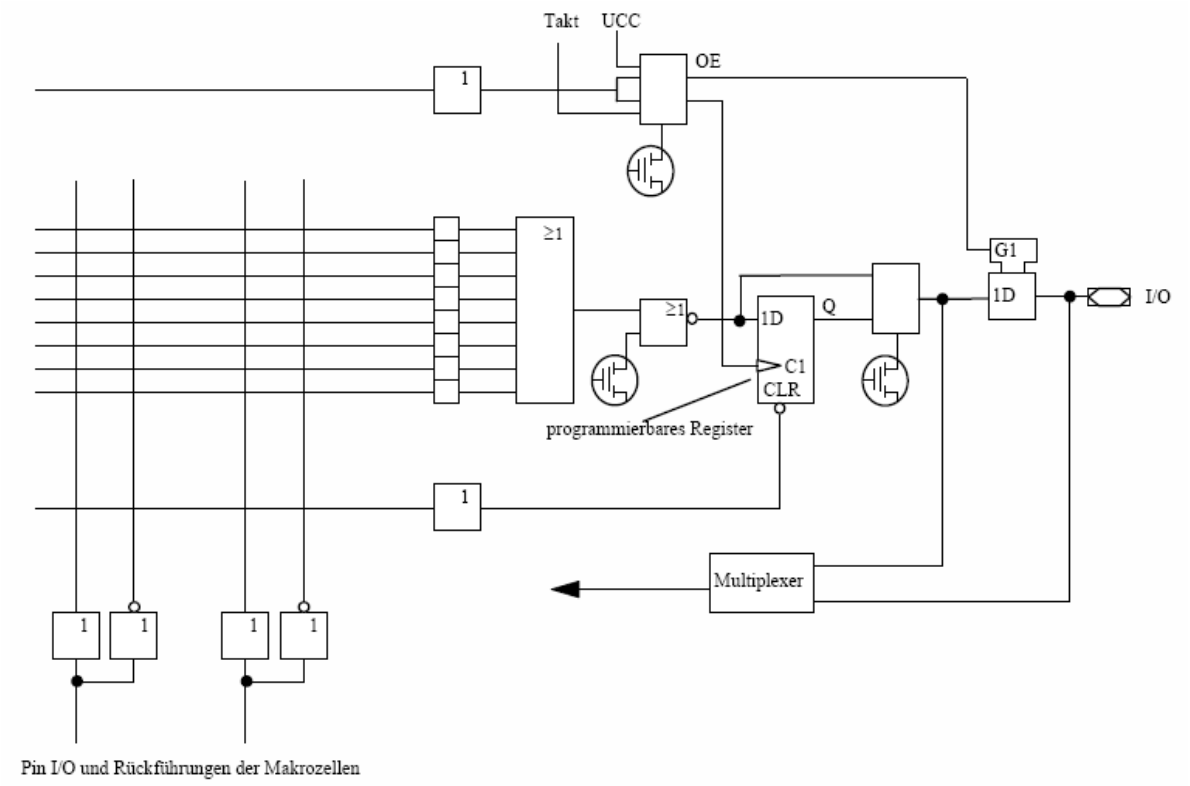

Bild 2.8: Struktur einer Makrozelle eines klassischen EPLDs.

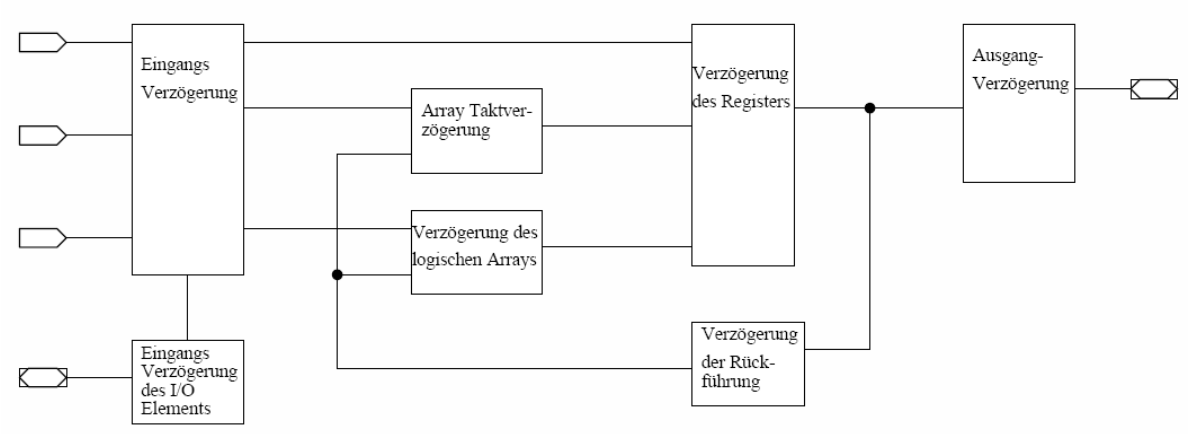

Bild 2.9: Schema der Signallaufzeiten zwischen Eingang und Ausgang eines klassischen EPLDs.

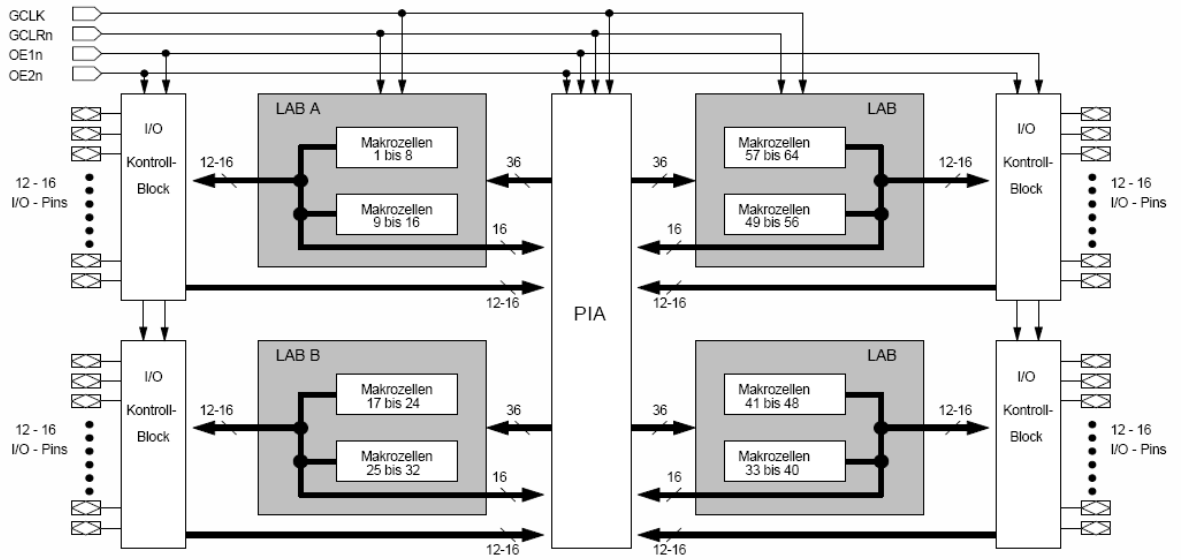

Bild 2.10: Aufbau eines MAX7064 EPLD-Bausteins.

Makrozellen können über ein „Programmable Interconnect Array“ (PIA) untereinander verbunden werden. Den schematischen Aufbau der beschriebenen Makrozelle zeigt Bild 2.11. Auch für diese EPLDs soll ein Übersichtsschema der zeitlichen Verzögerung der Signale zwischen Ein- und Ausgang angegeben werden. Gegenüber den klassischen EPLDs ist der Signallaufplan deutlich komplexer.

2.4 FPGA's

Die nächste Serie der programmierbaren Bausteine sind die FPGAs (**F**ield **P**rogrammable **G**ate **A**rrays). Als programmierbare Elemente werden hierbei statische RAM–Zellen eingesetzt. Zu dieser Gruppe von Bausteinen zählen alle LCAs (**L**ogic **C**ell **A**rrays) und die FLEX–Serie (**F**lexible **L**ogic **E**lement Matri**x**) der Firma Altera. Die Komplexität dieser Bauelemente reicht von einigen Hundert bis zu einigen Zehntausend Gatteräquivalenten. Diese Bausteine haben jedoch gegenüber den bisher beschriebenen einen Nachteil: sie verlieren beim Abschalten der Versorgungsspannung ihre Information und müssen deshalb nach jedem Einschalten neu programmiert werden. Dies kann direkt über eine Programmierschnittstelle erfolgen, oder durch ein sogenanntes Konfigurations–EPROM, in dem die Information fest gespeichert ist.

Der prinzipielle Aufbau dieser Bausteine soll anhand der FLEX 8000–Serie näher beschrieben werden. Zwischen den Ein–Ausgabe–Elementen an den Rändern des ICs verlaufen horizontal und vertikal durchgehende Bereiche mit Leitungen, über welche alle Verbindungen von und nach außen laufen. Zwischen den Bereichen mit den Leiterbahnen liegen sogenannte LABs (Logical Array Blocks), welche sich wiederum aus einzelnen logischen Elementen zusammensetzen. Ein Logic Array Block (LAB) besteht

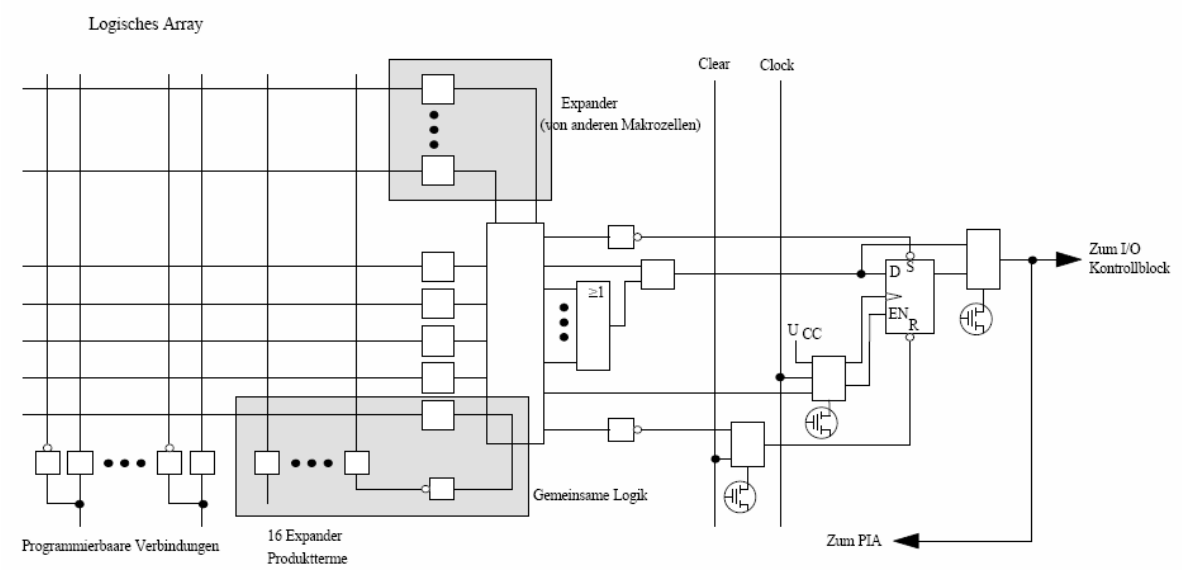

Bild 2.11: Struktur einer Makrozelle.

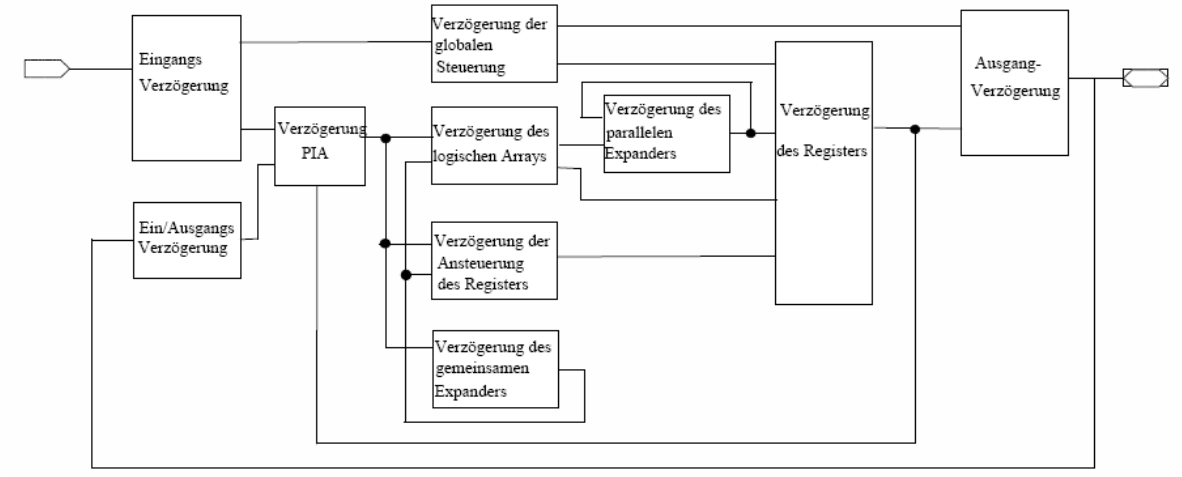

Bild 2.12: Schema der Signallaufzeiten zwischen Eingang und Ausgang eines EPLDs der Serie MAX7000.

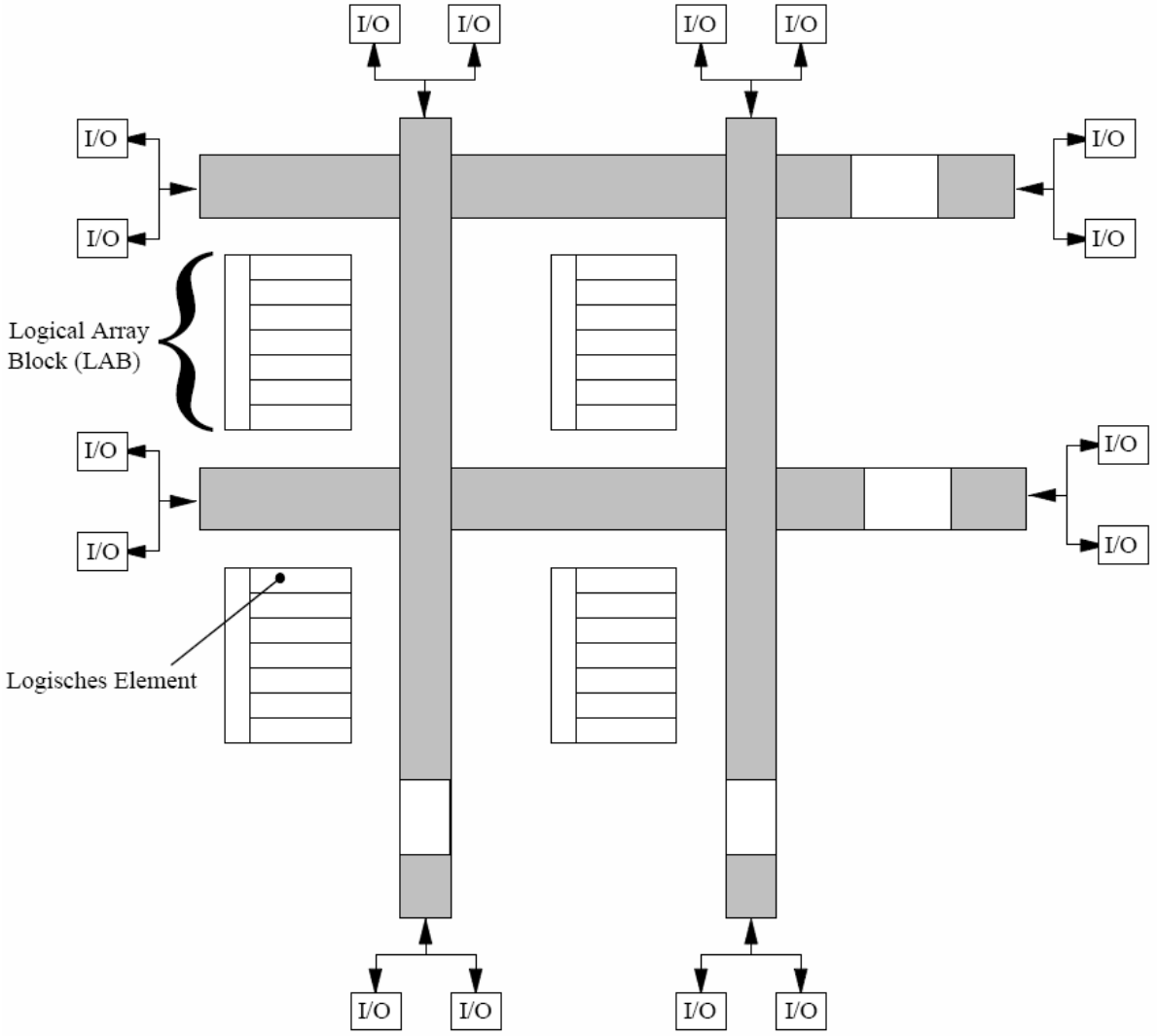

Bild 2.13: Blockschaltbild des Aufbaus der FLEX 8000 Serie.

aus acht logischen Elementen, den zugehörigen Carry und Cascade Pfaden, den LAB–Steuersignalen und einem lokalen Verbindungsblock. Diese Grundstruktur des Bauelements ist in Bild 2.13 schematisch dargestellt. Die Eingangsdaten für die Elemente werden entweder aus den horizontalen Verbindungsbereichen oder von speziellen Eingangspins direkt geliefert. Die Ausgänge der Logischen Elemente werden sowohl zu den horizontalen wie auch den vertikalen Verbindungsbereichen und zu den Eingangsverknüpfungen des LAB geführt.

Das Logikelement ist die kleinste logische Einheit eines FLEX 8000–Bausteins. Jedes dieser Elemente enthält eine „look–up table (LUT)„ mit 4 Eingängen, je ein Element für Übertrags- und Kaskadiersignale (Carry Chain, Cascade Chain) und ein programmierbares Flipflop, das als D–, T–, JK– oder SR–FF eingesetzt werden kann. Die LUT ist eine Art von Funktionsgenerator, der aus den 4 Eingangsvariablen jede logische Verknüpfung

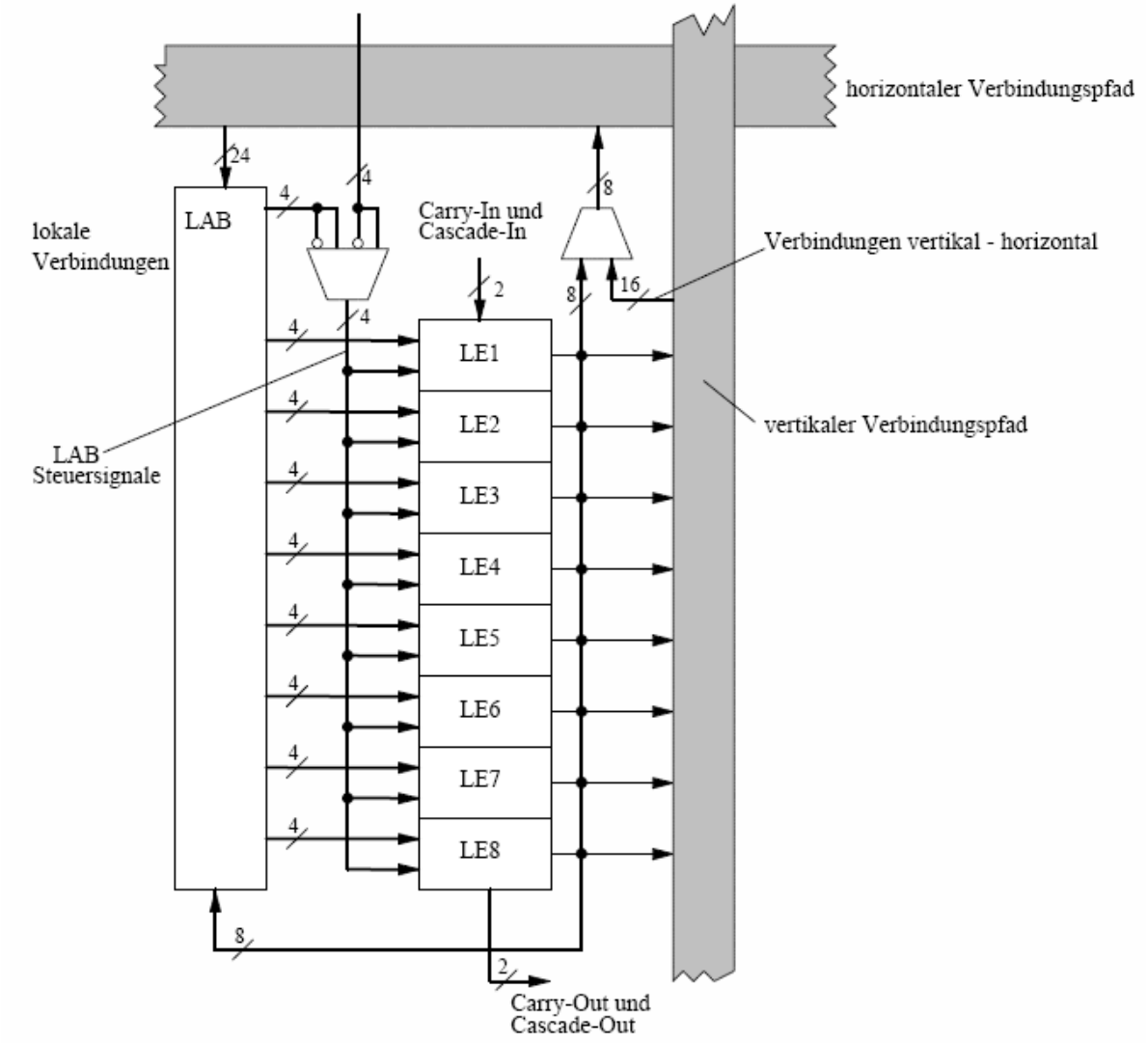

Bild 2.14: Schematischer Aufbaus eines Logic Array Blocks.

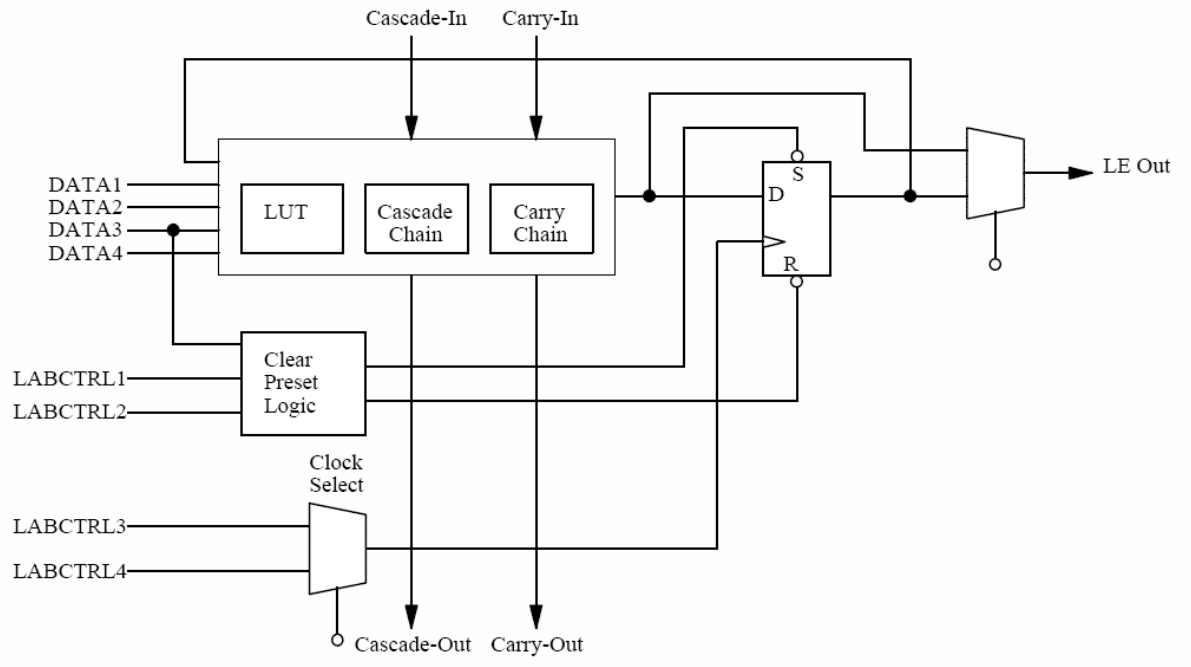

Bild 2.15: Blockschaltbild eines Logikelements.

erzeugen kann. Wird ein Logikelement ausschließlich für kombinatorische Logik eingesetzt, so wird der Ausgang der LUT am Flipflop vorbei, direkt zum Ausgang geführt. Die beiden Datenpfade Carry und Cascsade sind Hochgeschwindigkeitsverbindungen zwischen den einzelnen Logikelementen. Der Carry Pfad unterstützt die Entwicklung von Zählern mit hohen Taktfrequenzen und der Cascade Pfad implementiert breitgefächerte Eingangsfunktionen mit einem Minimum an Laufzeitverlusten. Ein Logikelement kann in vier verschiedenen Betriebsarten eingesetzt werden:

1. „Normaler Betrieb“,
2. „Arithmetische Betriebsart“,
3. „Vorwärts/Rückwärts Zähler“ und
4. „Rücksetzbarer Zähler“.

Die zugehörigen Signalpfade sind in Bild 2.16 dargestellt. Betriebsart 1 ist geeignet für allgemeine logische Anwendungen und für Dekodierschaltungen mit vielen Eingängen, die den Vorteil des Kaskadiersignals ausnutzen können. Hierbei werden die vier Dateneingänge und das Carry–In Signal als Eingangssignale für das Logische Element benutzt. Für das Carry–In Signal wird vom *MAX+PLUSII* Compiler immer der Eingang DATA3 ausgewählt.

2.4.1 Die Bausteinserie Cyclone II

Der Cyclone II Baustein gehört zur Familie der FPGAs (**F**ield **P**rogrammable **G**ate

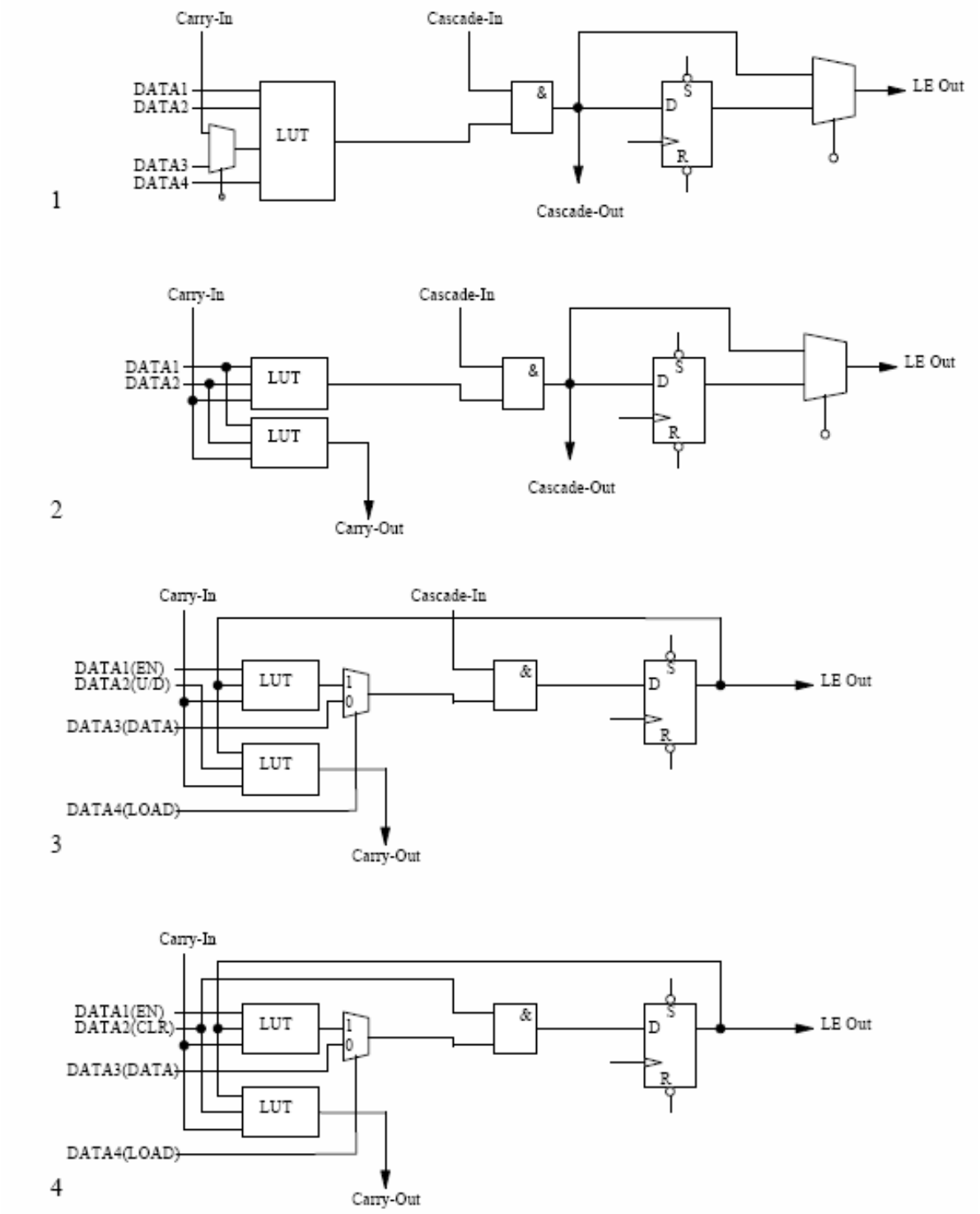

Bild 2.16: Signalverläufe bei den verschiedenen Betriebsarten eines Logischen Elements.

Kennzeichen	EP2C5	EP2C8	EP2C15	EP2C20	EP2C35	EP2C50	EP2C70
Logikelemente (LEs)	4608	8256	14448	18752	33216	50528	68416
M4K RAM blocks	26	36	52	52	105	129	250
Total RAM bits	119808	165888	239616	239616	483840	594432	1152000
Embedded multipliers	13	18	26	26	35	86	150
PLLs	2	2	4	4	4	4	4
Max. Benutzer I/O Pins	158	182	315	315	475	450	622

Tabelle 2.2: Bausteinübersicht der Serie Cyclone II.

Arrays). Als programmierbare Elemente werden hierbei statische RAM–Zellen eingesetzt. Zu dieser Gruppe von Bausteinen zählen alle LCAs (**L**ogic **C**ell **A**rrays), eben auch die Cyclone II von Altera. Die Komplexität dieser Bausteine reicht von einigen Tausend bis zu einigen Zehntausend Gatteräquivalenten. Eine kleine Baustainübersicht ist in Tabelle 2.2 dargestellt. Diese Bausteine haben jedoch gegenüber den PLDs (**P**rogrammable **L**ogic **D**evices) einen Nachteil. Sie verlieren beim Abschalten der Versorgungsspannung Ihre Informationen und müssen deshalb nach jedem erneuten Einschalten der Versorgungsspannung neu programmiert werden. Dies kann direkt über eine Programmierschnittstelle erfolgen, oder durch ein sogenanntes Konfigurations–EPROM, in dem die Informationen fest gespeichert sind.

2.4.1.1 Prinzipieller Aufbau der Cyclone II Serie

Zwischen den Ein-Ausgabe-Elementen an den Rändern des IC's verlaufen horizontal und vertikal durchgehende Bereiche mit Leitungen, über welche alle Verbindungen von und nach außen laufen. Zwischen den Bereichen mit den Leiterbahnen liegen sogenannte LABs (**L**ogical **A**rray **B**locks) und EABs (**E**mbedded **A**rray **B**lock). Die Logical Array Blocks setzen sich aus einzelnen logischen Elementen zusammen.

2.4.1.1.1 Logic Array Block (LAB)

Ein Logic Array Block (LAB) besteht aus acht logischen Elementen, den zugehörigen Carry und Cascade Pfaden, den LAB–Steuersignalen und einem lokalen Verbindungsblock. Diese Grundstruktur des Bauelements ist in Bild 2.18 schematisch dargestellt. Die Eingangsdaten für die Elemente werden entweder aus den horizontalen Verbindungsbereichen oder von speziellen Eingangspins direkt geliefert. Die Ausgänge der Logischen Elemente werden sowohl zu den horizontalen wie auch den vertikalen Verbindungsbereichen und zu den Eingangsverknüpfungen des LAB geführt.

2.4.1.1.2 Logikelement (LE)

Das Logikelement ist die kleinste logische Einheit eines ACEX 1K Bausteins. Jedes dieser Elemente enthält eine „Look Up Table (LUT)“ mit vier Eingängen, je ein Element für Übertrags- und Kaskadiersignale (Carry Chain, Cascade Chain) und ein programmierbares Flipflop, das als D–, T–, JK– oder RS–Flipflop eingesetzt werden kann.

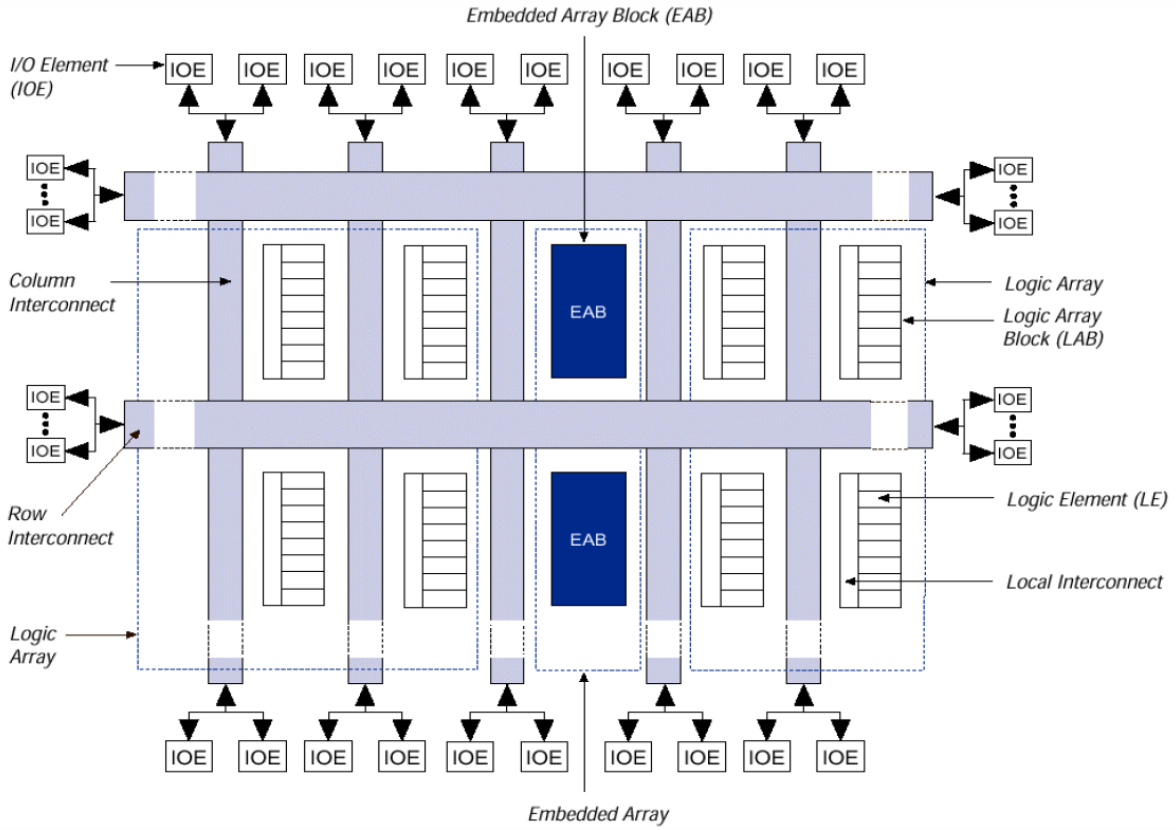

Bild 2.17: Blockschaltbild des Aufbaus der Cyclone II Serie.

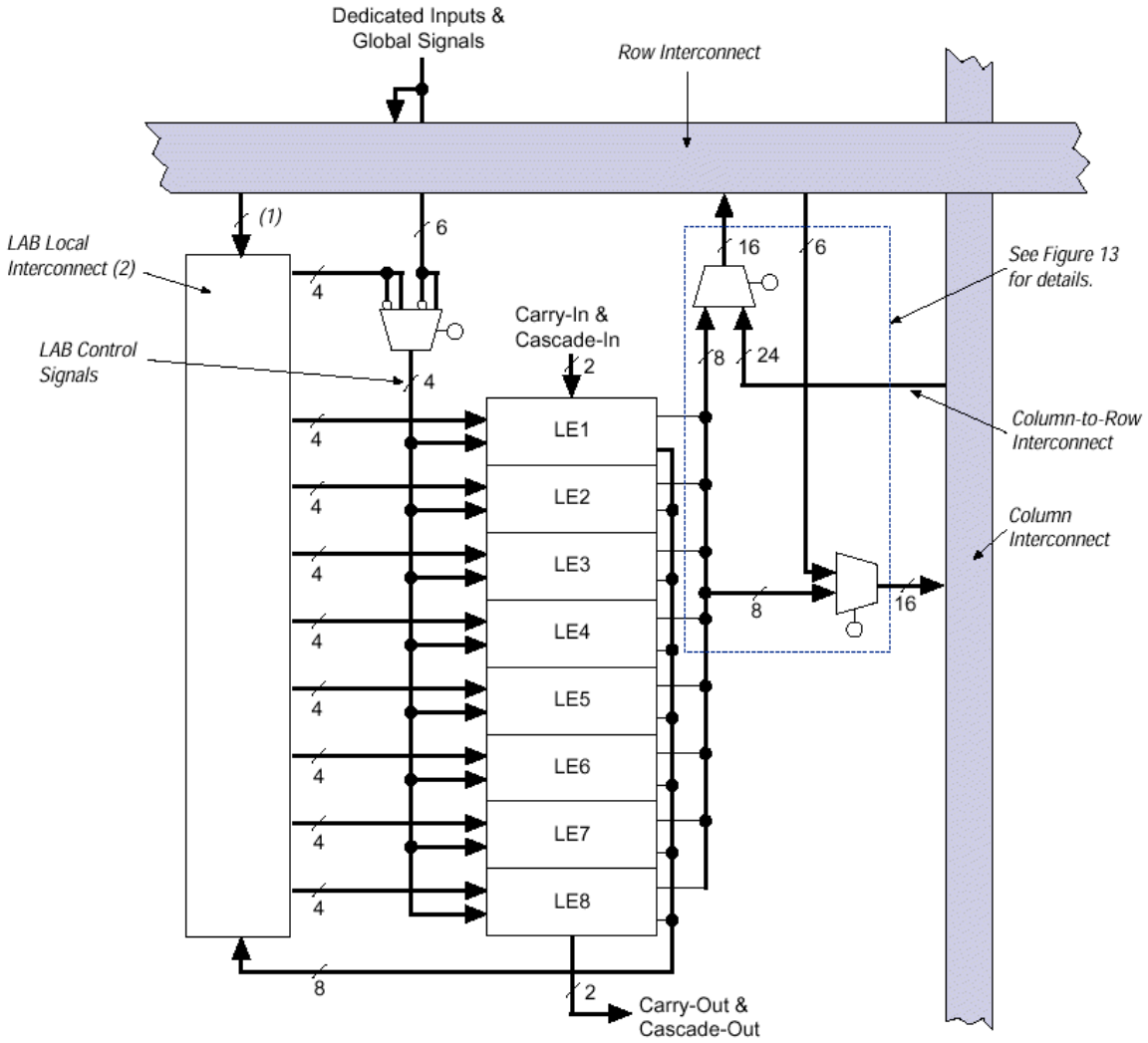

Bild 2.18: Schematischer Aufbau eines Logic Array Block.

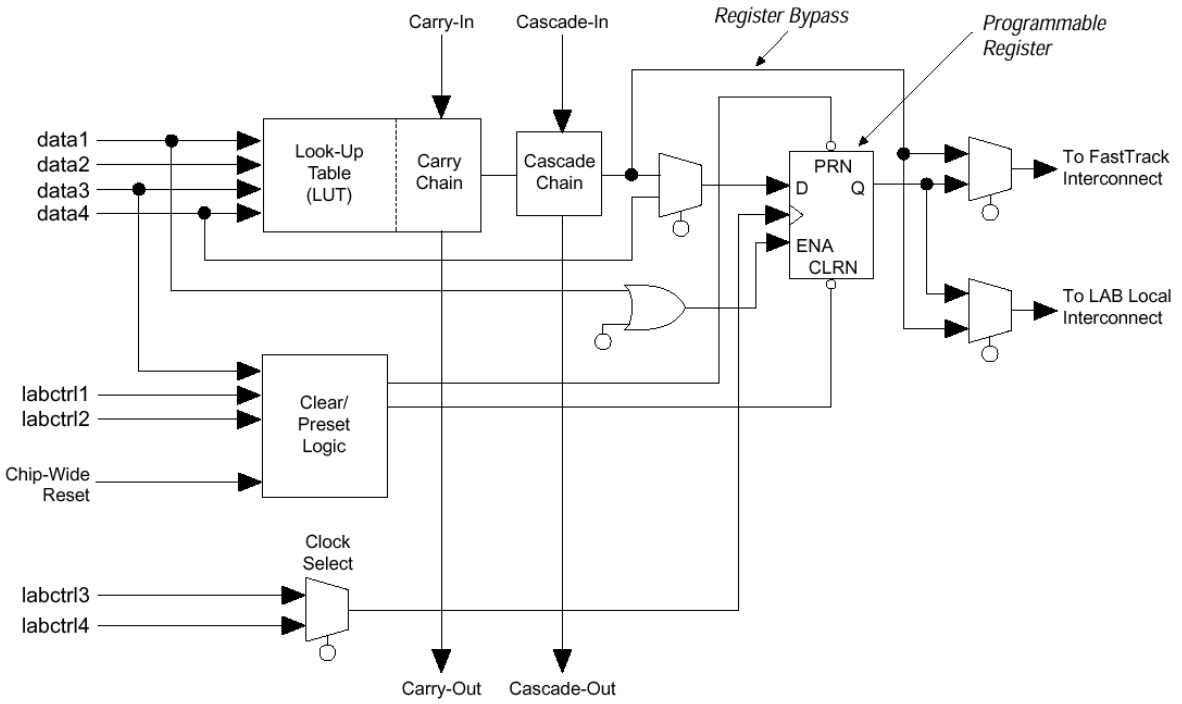

Bild 2.19: Schematischer Aufbau eines Logikelements.

Die LUT ist eine Art von Funktionsgenerator, der aus den vier Eingangsvariablen jede logische Verknüpfung erzeugen kann. Wird ein Logikelement ausschließlich für kombinatorische Logik eingesetzt, so wird der Ausgang des LUT am Flipflop vorbei, direkt zum Ausgang geführt. Die beiden Datenpfade Carry und Cascade sind Hochgeschwindigkeitsverbindungen zwischen den einzelnen Logikelementen. Der Carry Pfad unterstützt die Entwicklung von Zählern mit hohen Taktfrequenzen und der Cascade Pfad implementiert breitgefächerte Eingangsfunktionen mit einem Minimum an Laufzeitverlusten. Ein Logikelement kann in vier verschiedenen Betriebsarten eingesetzt werden:

1. „Normaler Betrieb“,
2. „Arithmetische Betriebsart“,
3. „Vorwärts/Rückwärts Zähler“ und
4. „Rücksetzbarer Zähler“.

„Normale Betriebsart“

Diese Betriebsart ist geeignet für allgemeine logische Anwendungen und für Dekodierschaltungen mit vielen Eingängen, die den Vorteil des Kaskadiersignals ausnützen. Hierbei werden die vier Dateneingänge und das Carry–In Signal als Eingangssignale für das logische Element benutzt.

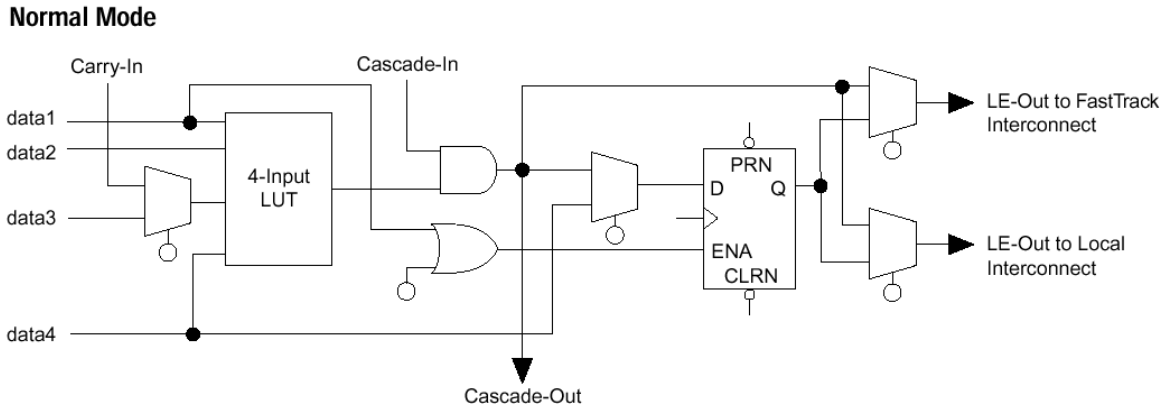

Bild 2.20: Schematischer Aufbau eines Logikelements in der Betriebsart 1.

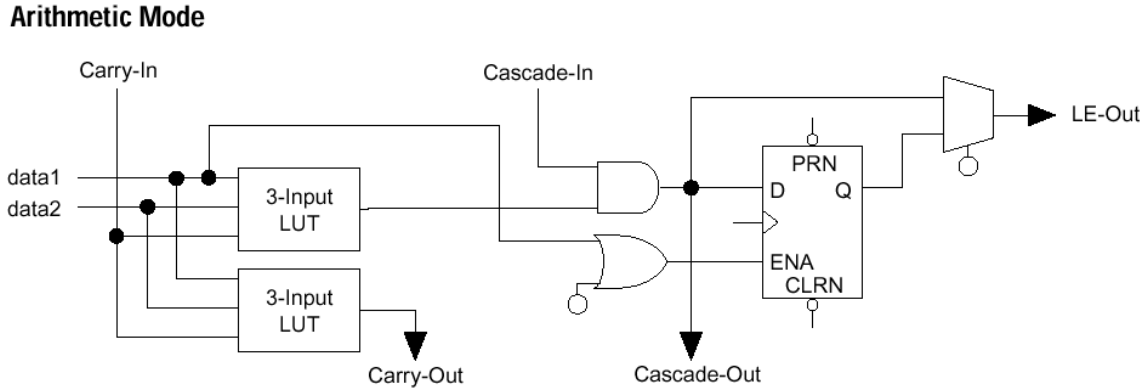

Bild 2.21: Schematischer Aufbau eines Logikelements in der Betriebsart 2.

„Arithmetische Betriebsart“

Dabei sind zwei LUT miteinander kombiniert. Aus dem ersten LUT, den zwei Eingangssignalen und dem Carry–In Signal wird das kombinatorische Ausgangssignal erzeugt. Diese Betriebsart eignet sich für Addierer, Akkumulatoren und Komparatoren, die die Vorteile des Carry–Out Signals ausnützen. Das Carry–Out Signal wird mit Hilfe des zweiten LUT und den drei gemeinsam genutzten Eingangssignalen erzeugt.

„Vorwärts/Rückwärts Zähler“

Diese Betriebsart nutzt zwei LUT's mit 3 Eingängen. Ein LUT erzeugt das Carry–Out Signal, das andere erzeugt das Zählsignal. Ein 2 zu 1 Multiplexer steuert das synchrone Laden eines Startwertes. Ein asynchrones Laden des Startwertes kann über die Signale Clear und Preset erfolgen. Zum Steuern des Vorwärts/Rückwärts Zählers stehen die Eingangssignale Counter Enable, Clock Enable, Up/Down, Data und nload zur Verfügung.

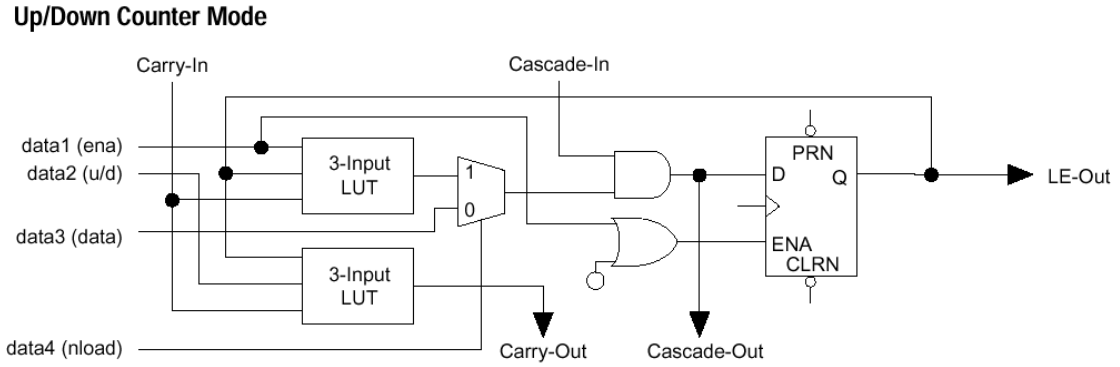

Bild 2.22: Schematischer Aufbau eines Logikelements in der Betriebsart 3.

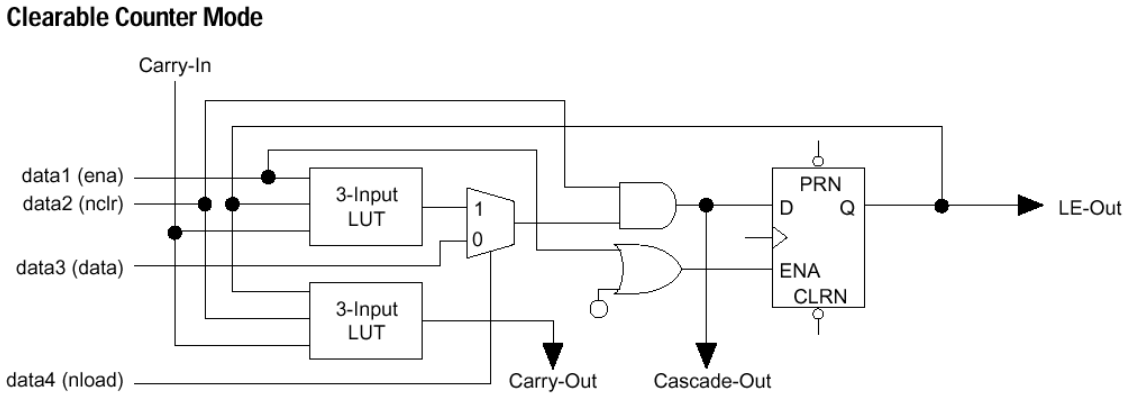

Bild 2.23: Schematischer Aufbau eines Logikelements in der Betriebsart 4.

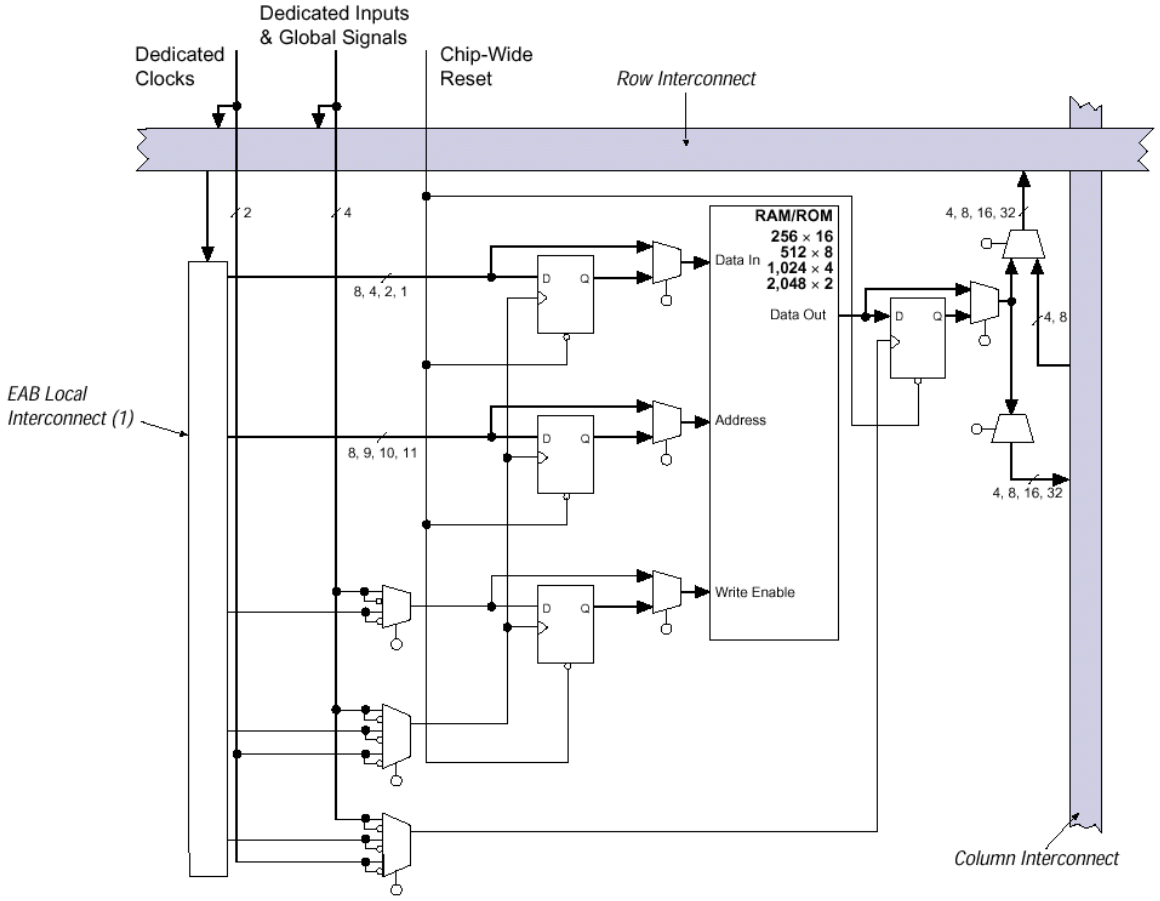

Bild 2.24: Konfiguration eines EAB als Single Port RAM.

2.4.1.1.3 Embedded Array Block

EABs sind aus Blöcken flexibler RAM Zellen aufgebaut, jedes EAB kann aus 8 Eingängen und 16 Ausgängen jede logische Funktion realisieren. Durch verschalten mehrerer EABs können komplexe Schaltungen, wie Multiplizierer, Microcontroller und State Machines aufgebaut werden. Hauptsächlich werden EABs jedoch zur Speicherung von Daten genutzt, dazu besitzt jedes EAB 4096 Bit, aus denen Speicher unterschiedlicher Typen aufgebaut werden kann:

- RAM
- ROM
- Dual Port RAM
- FIFO

Wie erwähnt, stehen jedem EAB Block 4096 Bit zur Verfügung, diese Bits können unterschiedlich auf Speichertiefe und Speicherbreite verteilt werden. Mögliche Einteilungen der Speicherbreite sind aus Bild 2.25 zu entnehmen. Auch bei Nutzung der EABs als

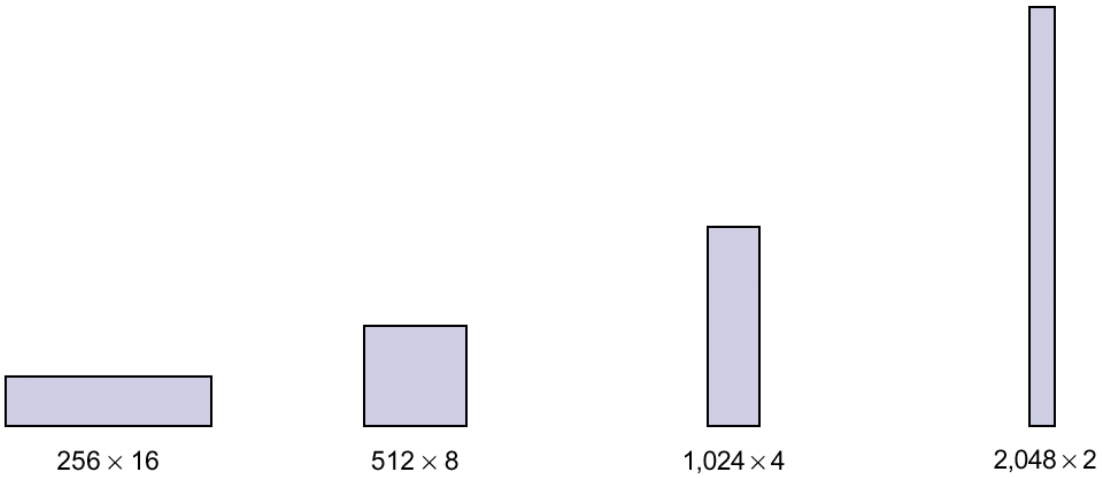

Bild 2.25: Mögliche Speicherbelegung eines EAB.

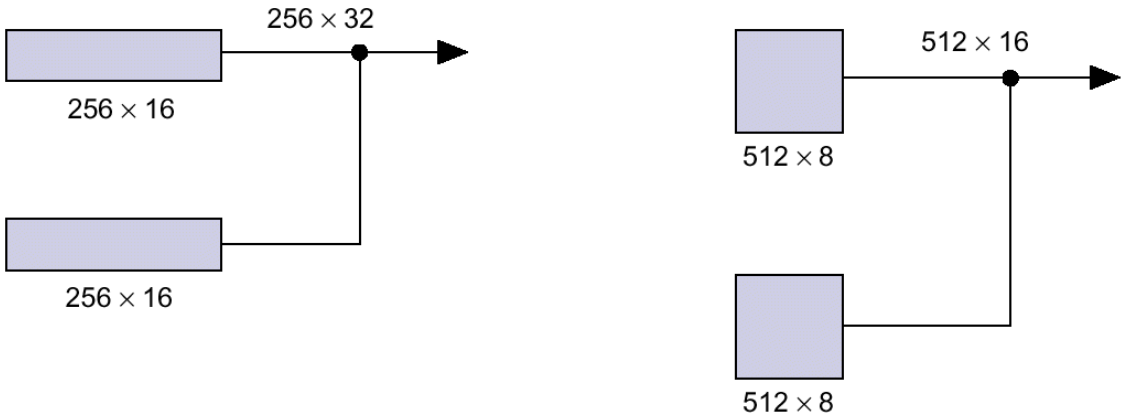

Bild 2.26: Verschaltung zweier EABs.

Speicher besteht die Möglichkeit mehrere EABs miteinander zu verknüpfen. Durch diese Verknüpfung kann die Speicherkapazität erhöht werden. Es können jeweils nur EABs mit gleicher Speicherbreite verknüpft werden. Durch diesen Umstand ist lediglich die Speichertiefe, nicht aber die Speicherbreite, durch Kombination mehrerer EABs änderbar.

3 Eingabe von Schaltungsinformationen

3.1 VHDL als Standardsprache

VHDL ist ein Ergebnis des „very high speed integrated circuit“ (VHSIC) Programmes, das vom amerikanischen Verteidigungsministerium in den späten siebziger Jahren initiiert wurde, um die Grundlagen für eine neue Generation schneller hochintegrierter Schaltkreise zu schaffen. VHDL steht für VHSIC Hardware Description Language und ist seit 1986 IEEE Standard. Die heute aktuelle Version von VHDL hat die Bezeichnung IEEE–1164. VHDL kann eine recht schwer zu erlernende Sprache sein, wenn man nur das VHDL Language Reference Manual als Grundlage benutzt und versucht, sich alle Definitionen und Vereinbarungen anzueignen. Um eine Schaltung zu definieren benötigt man jedoch nur sehr selten den gesamten Vorrat an Möglichkeiten, so daß es meistens ausreicht, mit den wichtigsten Elementen dieser Sprache vertraut zu sein.

Für das Praktikum ist ein Erlernen der Sprache nicht notwendig, da hier nur exemplarisch eine kleinere Schaltung mit VHDL erstellt werden soll, um einen Vergleich mit den anderen Eingabemöglichkeiten zu haben. Aus diesen Grund ist der Einsatz von VHDL an dieser Stelle nur anhand einiger einfacher Beispiele aufgezeigt, was für die Bearbeitung des Versuches 1.3 ausreichend sein sollte. Als ausführliche Einführung ist das Buch VHDL von Douglas L. Perry, Mc Graw Hill Verlag zu empfehlen. Anhand von einfachen Beispielen sollen nun die grundlegenden Elemente der Sprache kurz erklärt werden. Alle Schlüsselwörter der Sprache VHDL sind in Großbuchstaben dargestellt. Zuerst soll als einfaches Beispiel ein D–Flipflop mit Enable–Eingang erläutert werden. Das D–Flipflop besitzt die folgenden Ein- und Ausgänge: Eine VHDL Beschreibung für

D	-	Signaleingang
CLK	-	Takteingang
CLRN	-	Clear Not Eingang
ENA	-	Enable Eingang
Q	-	Ausgang

dieses Flipflop könnte so aussehen:

```
LIBRARY ieee; -- Bibliotheksvereinbarung
USE ieee.std_logic_1164.all;

ENTITY dff IS -- Schaltungssymbol
      PORT
        (
            d    : IN STD_LOGIC;
```

```
            clk  : IN STD_LOGIC;
            clrn : IN STD_LOGIC;
            ena  : IN STD_LOGIC;
            q    : OUT STD_LOGIC
         );
END dff;

ARCHITECTURE a OF dff IS -- Schaltungsbeschreibung
      SIGNAL tmp : STD_LOGIC;

BEGIN
      PROCESS (clk,clrn)
      BEGIN
            IF clrn = '0' THEN
              tmp <= '0';
ELSIF (clk'EVENT AND clk = '1') THEN
              IF ena = '1' THEN
                tmp <= d;
            ELSE
                tmp <= tmp;
              END IF;
            END IF;
      END PROCESS;
        q <= tmp;
END a;
```

In den ersten beiden Zeilen sind die verwendeten Bibliotheken aus dem ALTERA–Entwicklungssystem beschrieben. Das Element IEEE.STD_LOGIC_1164.ALL aus der Bibliothek LIBRARY IEEE enthält alle Funktionen die für die Bearbeitung des Versuches 1.3 notwendig sind. In VHDL muß zuerst das Schaltungssymbol mit den Ein- und Ausgängen definiert werden. Dazu wird das Element ENTITY verwendet. Dieses Element ist die grundlegende Beschreibung eines Schaltungsentwurfs (building block). Die höchste Ebene des Entwurfs wird als top–level ENTITY bezeichnet. Wenn der Entwurf hierarchisch aufgebaut ist, wird die top–level Beschreibung lower–level Beschreibungen enthalten. Diese lower–level Beschreibungen werden dann auch als lower–level ENTITY bezeichnet.

Anschließend muß das Verhalten dieses Flipflops beschrieben werden. Die Schaltungsbeschreibung ist in der Zeile ARCHITECTURE A OF DFF IS definiert. Diese Schaltungsbeschreibung ist mit dem Namen „a“ bezeichnet, da VHDL verschiedene Beschreibungen für ein Element zuläßt. Der Anfang dieser Beschreibung ist durch BEGIN und das Ende ist durch END A gekennzeichnet. Nach der ARCHITECTURE–Vereinbarung ist die lokale Variable TMP als Signal definiert, die identisch mit dem Ausgangssignal Q ist. Die Definition dieser Variablen ist notwendig, da während des Programmablaufes

auf Ausgangssignale nicht zugegriffen werden kann. Der Ausgangzustand des Flipflops wird daher in dieser lokalen Variable gespeichert und am Ende der ARCHITECTURE–Vereinbarung dem Ausgangssignal Q zugewiesen. Das ARCHITECTURE–Statement beinhaltet hier nur ein Statement, dass PROCESS–Statement. Dieses wiederum beinhaltet eine Reihe von Elementen:

- die *sensitivity list*,
- den *process declarative part* und
- den *statement part*.

In unserem Beispiel ist die *sensitivity list* der Teil, der nach dem Statement PROCESS in den Klammern steht, also *clk* und *clrn*. Der zweite Teil, der *process declarative part* ist der Bereich zwischen der *sensitivity list* und dem Wort BEGIN. Hier werden lokale Variable und Konstante definiert, die innerhalb des PROCESS–Statements benötigt werden, was in diesem Beispiel nicht der Fall ist. Der dritte Teil, der *statement part*, ist der gesamte Bereich zwischen BEGIN und END PROCESS. Alle Befehle in diesem Bereich werden wie bei allen Programmiersprachen sequentiell ausgeführt, d.h. die Reihenfolge der Befehle bestimmt die Reihenfolge der Ausführung. Innerhalb des *statement parts* stehen in diesem Beispiel nur IF (logische Bedingung) THEN–Anweisungen, die für dieses Beispiel vollkommen ausreichend sind. IF–STATEMENTS sollten immer so angeordnet werden, daß sich ein eindeutiger Entscheidungsbaum ergibt. Die Zuweisung eines logischen Pegels ('0' oder '1') bzw. des Wertes eines anderen Signales erfolgt über den Operator <=. Der Ausdruck CLK'EVENT AND CLK = '1' bezeichnet die ansteigende Flanke des *clk*–Signales. Neben dem IF–STATEMENT stehen natürlich noch viele andere Anweisungen zur Verfügung, von denen hier nur die CASE– und die LOOP–Anweisung erwähnt werden sollen. Ein CASE–STATEMENT erlaubt die Verzweigung bezüglich einer logischen Bedingung und kann für die behandelten Beispiele auch durch mehrere geschachtelte IF–STATEMENTS ersetzt werden.

```
CASE (expression) IS
      WHEN constant_value1 =>
           statement a;
           statement b;
      WHEN constant_value2 =>
           statement c;
           statement d;
      WHEN OTHERS =>
           statement e;
           statement f;
END CASE;
```

Die LOOP–Anweisung ermöglicht es, Befehle innerhalb einer Schleife für verschiedene Werte einer Variablen nacheinander durchzuführen. Sie ist mit der for Anweisung der Programmiersprache PASCAL vergleichbar.

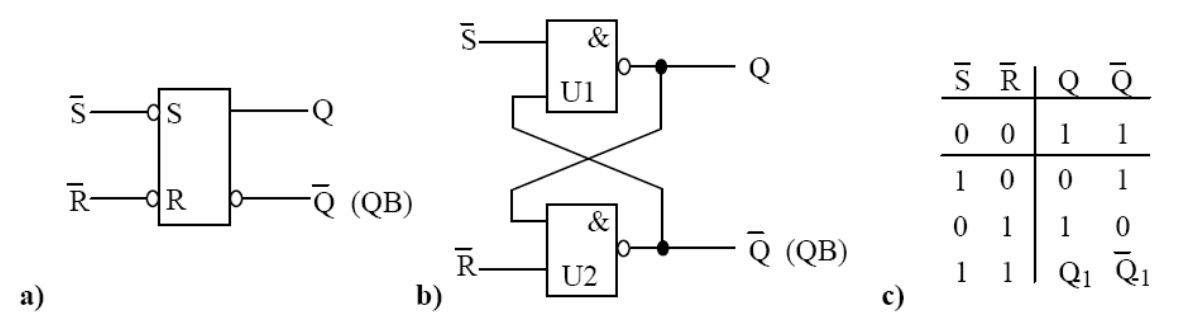

Bild 3.1: RS–Flipflop mit NAND–Gattern.

```
Label1:
FOR i IN 1 to 10 LOOP
    statement1;
    statement2;
END LOOP Label1;
```

VHDL erlaubt es auch, das zeitliche Verhalten einer Schaltung zu modellieren, z.B. durch Angabe der Gatterlaufzeiten. Eine solche Anweisung sieht für ein AND–Gatter mit den Eingängen A, B und dem Ausgang C folgendermaßen aus:

```
C <= A AND B AFTER 2ns;
```

Die Gatterlaufzeit wird hier durch das AFTER–STATEMENT auf 2 ns festgelegt. Diese Anweisung sollte allerdings innerhalb des ALTERA–Entwicklungssystems nicht angewendet werden, da die Gatterlaufzeiten durch den verwendeten Baustein bzw. der Umsetzung durch den Compiler vorgegeben werden. Neben dem im Beispiel oben definierten Standardsignalen (STD_LOGIC) erlaubt VHDL eine Reihe weiterer Variablentypen, z.B. BIT, INTEGER, REAL etc. Es können auch Signale zu Vektoren zusammengefaßt werden. So wird z.B. mit

```
Q :OUT STD_LOGIC_VECTOR(1 TO 4)
```

ein Ausgabevektor mit 4 Elementen definiert. Die Elemente können dann mit Q[1], Q[2] usw. ausgewählt werden. Als zweites Beispiel soll ein RS–Flipflop aus NAND–Gattern besprochen werden. Die logische Funktion der Schaltung läßt sich durch die Wahrheitstabelle (Bild 3.1.c) beschreiben. Um die Schaltung später beschreiben zu können, sind die beiden NAND–Gatter mit U1 und U2 gekennzeichnet. Eine mögliche Beschreibung dieses Flipflops könnte in VHDL so aussehen:

```
LIBRARY ieee; -- Bibliotheksvereinbarung
USE ieee.std_logic_1164.all;

ENTITY rsff IS -- Schaltungssymbol
    PORT ( set , reset : IN STD_LOGIC;
```

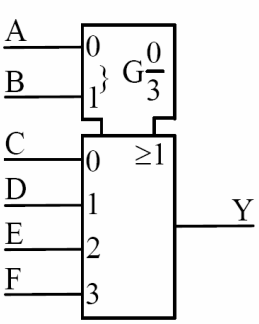

Bild 3.2: 4–zu–1 Multiplexer, logisches Schaltsymbol.

```
          q , qb : OUT STD_LOGIC );
END rsff;

ARCHITECTURE a OF rsff IS -- Schaltungsbeschreibung
signal tmp1 : STD_LOGIC := '1';
signal tmp2 : STD_LOGIC := '0';

function und_nicht(a,b: STD_LOGIC) RETURN STD_LOGIC IS
variable c: STD_LOGIC;
BEGIN
  c := NOT(A AND B);
RETURN c;
END und_nicht;

BEGIN
PROCESS (set,reset)
BEGIN
  tmp1 <= und_nicht(tmp2,NOT(set));
  tmp2 <= und_nicht(tmp1,NOT(reset));
END PROCESS;
q <= tmp1;
qb <= tmp2;
END a;
```

Beachten Sie, daß im Gegensatz zu Bild 3.1 die nicht negierten Eingangssignale set und reset verwendet werden. Hier wurden die NAND–Gatter als Unterfunktion UND_NICHT realisiert, um die Verwendung von Unterfunktionen zu erläutern. Die Deklaration FUNCTION UND_NICHT(A,B: STD_LOGIC) RETURN STD_LOGIC IS bedeutet, daß die Unterfunktion UND_NICHT mit den Variablen A und B vom Typ STD_LOGIC aufgerufen wird und als Ergebnis eine Variable vom Typ STD_LOGIC zurückliefert. Als letztes Beispiel soll ein 4–zu–1 Multiplexer beschrieben werden, dessen Schaltsymbol und Wahrheitstabelle in Bild 3.2 bzw. in Tabelle 3.1 angegeben sind. Die Beschreibung

B	A	Y
0	0	C
0	1	D
1	0	E
1	1	F

Tabelle 3.1: 4–zu–1 Multiplexer, Wahrheitstabelle

dieses Bauelements in VHDL könnte folgendes Aussehen haben:

```
LIBRARY ieee; -- Bibliotheksvereinbarung
USE ieee.std_logic_1164.all;

Entity mux41 IS
   PORT (c, d, e, f, a, b : IN STD_LOGIC;
         y: OUT STD_LOGIC);
   END mux41;

ARCHITECTURE a OF mux41 IS
BEGIN
   PROCESS ( c, d, e, f, a, b )
VARIABLE muxval : INTEGER RANGE 0 TO 3;
BEGIN
  muxval := 0;
  IF ( a = '1' ) THEN
     muxval := muxval + 1;
  END IF;
  IF ( b = '1' ) THEN
     muxval := muxval + 2;
  END IF;
  CASE muxval IS
     WHEN 0 =>
        y <= c;
     WHEN 1 =>
        y <= d;
     WHEN 2 =>
        y <= e;
     WHEN 3 =>
        y <= f;
     WHEN OTHERS =>
        NULL;
  END CASE;
END PROCESS;
END a;
```

3.2 AHDL (Altera Hardware Description Language)

Wie VHDL ist auch AHDL eine Sprache, und logische Schaltungen in Form eines „Programms“ zu beschreiben. Von der Struktur her ist es vergleichbar mit der Programmiersprache C. Wie jede Eingabeform von Schaltungsinformation unterliegt auch die Beschreibung in AHDL bestimmten Regeln. Diese sind in aller Ausführlichkeit im MAX+PLUS® II AHDL–Handbuch nachzulesen. Wie bereits für VHDL gilt auch für AHDL, daß für das Praktikum ein Erlernen der Sprache nicht notwendig ist, da hier nur exemplarisch eine kleinere Schaltung mit AHDL erstellt werden soll, um einen Vergleich mit den anderen Eingabemöglichkeiten zu haben. Aus diesen Grund ist der Einsatz von AHDL an dieser Stelle nur anhand einiger einfacher Beispiele aufgezeigt.

Alle Schlüsselwörter der Sprache AHDL sind in Großbuchstaben dargestellt. Die reservierten Schlüsselwörter umfassen unter anderem alle logischen Funktionen, Flipflops und den logischen Anfangszustand X. Im MAX+PLUS® II–Programm sind eine Reihe von Eingabeschablonen für die Elemente von AHDL vorhanden. Unter Zuhilfenahme dieser „Templates“ kann mit wenig Aufwand eine komplexe digitale Schaltung entworfen werden. Nachfolgend das Beispiel eines 4–zu–1 Multiplexers mit der Funktionsbeschreibung in einer CASE–Struktur.

```
SUBDESIGN mux_case
(
       di[3..0]        : INPUT;     --Datenvektor
       adr[1..0]       : INPUT;     --Adressvektor
       muxaus          : OUTPUT;
)
BEGIN
CASE adr[] IS
   WHEN 0 =>
          muxaus=di[0];
   WHEN 1 =>
          muxaus=di[1];
   WHEN 2 =>
          muxaus=di[2];
   WHEN 3 =>
          muxaus=di[3];
END CASE;
END;
```

4 Kurzeinführung in Altera Quartus II

Die Entwicklungsumgebung Quartus II von Altera bietet alle Werkzeuge für den gesamten Ablauf eines komplexen Schaltungsentwurfs für programmierbare Logikbausteine. Von der Schaltungseingabe über die Synthese, die Simulation, die Programmierung bis zur Verifikation sind alle Elemente des Entwurfablaufs nahtlos im Programm integriert. Die Benutzerschnittstelle stellt sich als normale Windows–konforme Bedienoberfläche dar. Nach dem Starten des Programms öffnet sich das Hauptfenster mit den individuell anpassbaren Symbolleisten und Andockfenstern. Die Andockfenster können in der Symbolleiste ein- und ausgeschaltet werden. Die Bedeutung der einzelnen Schaltflächen wird beim Überfahren mit dem Mauszeiger eingeblendet. Neben den von anderen Programmen bekannten Menütiteln, wie File (Datei), Edit (Bearbeiten), View (Ansicht), Window (Fenster) und Help (Hilfe) gibt es die für Quartus spezifischen Menüs Project, Assignments und Processing. Die Bedeutung der Menüeinträge ist weitestgehend selbsterklärend. Neben der folgenden Kurzeinführung steht auch das Handbuch *Introduction to Quartus II* zur Verfügung.

4.1 Projektverwaltung

Prinzipiell sollte für jeden eigenen Teilentwurf, und sei er noch so klein, ein eigenes Projekt mit *eindeutigem* und *selbsterklärendem* Namen (z.B. fir_filter) angelegt werden. Ein Projekt wird begonnen, indem es neu erstellt wird. Dies geschieht unter dem Menüeintrag **File–New Project Wizard**. In den sich dann öffnenden Dialogfenstern werden folgende Projekteinstellungen vorgenommen:

1. Angabe des Projektverzeichnisses (zur Vermeidung späterer Probleme sollte für jedes Projekt ein eigenes Verzeichnis angelegt werden), Angabe des Projektnamens und des Namens der obersten Hierarchiestufe des Entwurfs (diese sind normal identisch mit dem Verzeichnisnamen).

2. Angabe von Dateien, die dem Projekt hinzugefügt werden (für einfache Projekte keine).

3. Spezifizieren von externen Entwurfsprogrammen (trifft für das Praktikum nicht zu).

4. Angabe der Bausteinfamilie (hier: Cyclone II und Zuweisung eines spezifischen Bausteins auswählen).

5. Auswahl des Bausteins, im Praktikum wird der Typ EP2C70F672C6N verwendet.

6. Übersicht der Einstellungen.

Für Unterentwürfe ist ein Unterverzeichnis anzulegen, welches als User Library eingetragen wird (über **Assignments–Settings–User Libraries**). Dorthin kopiert man alle Dateien von eigenen Unterentwürfen. Werden sogenannte Megafunctions eingebunden, so sollten diese mit dem Projektnamen als Präfix und einem folgenden eindeutigen Bezeichner benannt (z.B. FIR_FILTER_MULTIPLIZIERER) und ebenfalls im Unterordner abgelegt werden. Der Namenspräfix ist wichtig, damit es in übergeordneten Projekten keine Teile mit identischem Namen gibt. Alle Unterentwürfe in einem Unterverzeichnis abzulegen hat Vorteile, wenn das Projekt in ein übergeordnetes Projekt eingebunden werden soll. In diesem Fall genügt es, die Dateien des Unterverzeichnisses und die Entwurfsdateien eines Projekts in das Unterverzeichnis des übergeordneten Projekts zu kopieren. Für jedes Projekt kann ein Symbol angelegt werden. Außer bei VHDL–Projekten, die eigene Packages verwenden, können die Symbole automatisch erstellt werden, ansonsten müssen sie manuell erstellt werden.

Beim Erstellen eines Projekts legt Quartus neben der Projektdatei mit der Dateiendung *.qpf („Quartus Project File“) noch andere Dateien im Projektverzeichnis an, in denen die verschiedensten Einstellungen gespeichert sind. Ein bereits erstelltes Projekt kann unter dem Menüeintrag **File–Open Project** geöffnet werden. Nachdem ein Projekt erstellt oder geöffnet ist, sind links im Project Navigator die Hierarchie, die verwendeten Dateien (Files) und die Entwurfseinheiten (Design Units) aufgelistet. Der Project Navigator, wie auch andere Utility Fenster, können über die Utility Windows Symbolleiste oder über das Menü **View–Utility Windows** ein- und ausgeblendet werden. Es empfiehlt sich die Zoom Symbolleiste ebenfalls zur Symbolleistenfläche hinzuzufügen (über das Menü **Tools–Toolbars**). Das Projekt kann nun bearbeitet werden. Unter dem Menüeintrag **File–Close Project** wird das aktuelle Projekt geschlossen. Änderungen der Einstellungen werden implizit gespeichert. Ein kompletter Schaltungsentwurf beinhaltet verschiedene Dateien. Diese Dateien sind von der Bearbeitung her nebeneinander gleichberechtigt, werden jedoch durch den Zusammenhang im Entwurf hierarchisch strukturiert. Eine Datei, die zum Projekt gehören soll, kann unter dem Menüpunkt **File–New** oder über die Symbolleiste erstellt werden. Nach dem Auswählen des Menüpunktes erscheint ein Fenster, in dem die Art der zu erstellenden Datei ausgewählt werden kann. Zum Schaltungsentwurf stehen dabei die graphische Eingabe oder die Texteingabe zur Verfügung. Die für das Praktikum wesentlichen Dateitypen sind:
Unter dem Reiter „Device Design Files“:

- Block Diagram/Schematic File mit der Dateiendung *.bdf.
- VHDL File mit der Endung *.vhd.
- AHDL File mit der Endung *.tdf.

Unter dem Reiter „Other Files“:

- Block Symbol File mit der Dateiendung *.bsf.

- Vector Waveform File mit der Endung *.vwf.

Nach dem Öffnen oder Neuerstellen einer Datei erscheint auch die zum Bearbeiten erforderliche Symbolleiste. Es können mehrere Dateien geöffnet und bearbeitet werden, die dann in einzelnen Unterfenstern im Arbeitsbereich von Quartus II erscheinen. Durch Aktivieren der einzelnen Unterfenster wird die spezifische Symbolleiste umgeschaltet. Die bearbeiteten Dateien können unter dem Menüpunkt **File–Save** oder über das Diskettensymbol auf der allgemeinen Symbolleiste gespeichert werden. Dabei wird bei mehreren geöffneten Dateien jeweils nur die aktuell aktive gespeichert. Beim Schließen des Projekts oder beim Beenden des Programms erfolgt eine automatische Abfrage für jede geöffnete und veränderte Datei, ob diese gesichert werden soll.

4.2 Grafische Eingabe (Block Diagram/Schematic)

Der grafische Editor dient zur Eingabe eines Schaltplans. In der zugehörigen Symbolleiste sind die Werkzeuge dargestellt, die zur Eingabe von Schaltungen eingesetzt werden können. Das Symbol Tool lässt sich auch durch Doppelklick in die freie Fläche des Editors aktivieren. Dabei öffnet sich ein Fenster, in dem selbst erstellte (unter dem Ordner Project) oder vordefinierte Symbole aus den mit Quartus II mitgelieferten Bibliotheken ausgewählt und in die zu erstellende Schaltung eingefügt werden können. Bei bekanntem Symbolnamen kann man diesen auch direkt im Auswahlfenster eingeben. Mit den verschiedenen Verbindungswerkzeugen (**Node Tool**, **Bus Tool** und **Conduit Tool**) lassen sich die eingefügten Elemente miteinander verbinden. Die wohl wichtigsten Elemente sind Ein- und Ausgänge, sie stehen unter den Namen *input* und *output* zur Verfügung.

4.2.1 Erstellen von Symbolen (Block Symbol)

Zur hierarchischen Strukturierung eines Entwurfs können Schaltungsteile zu eigenen Symbolen zusammengefasst werden. Dies kann man automatisiert vornehmen, indem der Menüpunkt **File–Create/Update–Create Symbol Files for Current File** gewählt wird. Bei komplexeren Schaltungen mit z.B. mehrdimensionalen Bussen ist das integrierte Werkzeug jedoch überfordert. Dann muss das Symbol von Hand erstellt werden. Dies geschieht durch öffnen einer Block Symbol Datei. Im grafischen Editor sieht man den Rahmenkonstrukt eines Symbols. Durch Anklicken der grünen Umrandung kann man Ein- und Ausgänge hinzufügen. Das Symbol muss unter identischem Namen, wie die Entwurfsdatei für die es stehen soll, abgespeichert werden (**File–Save As**)(mit der Endung *.bsf, dem so genannten „Block Schematic File“).

4.3 Erstellen von Textdateien (AHDL File oder VHDL File)

Neben der grafischen Eingabe von Schaltungen besteht die Möglichkeit des formalen Entwurfs mit den Beschreibungssprachen AHDL und VHDL. Hierzu wird eine Textdatei ebenso wie die vorigen Dateitypen erstellt und geöffnet. Beim Bearbeiten stehen unter

dem Menüpunkt **Edit–Insert Template** verschiedene Rahmengerüste für Anweisungskonstrukte unter AHDL und VHDL zur Verfügung.

4.4 Erstellen von Simulationsstimuli (Vector Waveform File)

Für die Simulation der erstellten Schaltungen muss ein Stimulus erstellt werden. Dazu wird eine so genannte Vector Waveform Datei erstellt oder geöffnet. Im linken Bereich kann durch Doppelklicken und Auswählen des Node Finder die im Entwurf erstellten und nach dem Compilieren vorhandenen Eingänge, Ausgänge und Knoten eingefügt werden. In der zugehörigen Symbolleiste stehen verschiedene Werkzeuge zum Erstellen von Takten, Pulsen und Zählungen zur Verfügung. Der Zeitumfang für die Simulation wird unter dem Menüpunkt **Edit–End Time** eingestellt. Die Datei muss vor der Simulation gespeichert werden. Der Dateiname ist dabei automatisch vorgegeben. Für hierarchische Schaltungsentwürfe mit mehreren Entwurfsdateien muss er allerdings mit dem Namen der zu simulierenden Einheit übereinstimmen.

4.4.1 Einrichtung der VWF-Editor ab Quartus II v11.1

1. Eine Verknüpfung der Datei „$C:\altera\11.1\quartus\bin\quartus\_sh$“ in Desktop Erstellen.
2. In die Zielfenster der Verknüpfung muss $--$qsim als Argument hinzugefügt werden.
3. Den Laufenden Projekt öffnen und eine Input Datei erstellen oder öffnen.
4. Nach dem die Stimulus fertig modifiziert wurde kann die Simulation direkt gestartet werden. Der Projekt muss nicht wieder Kompiliert werden, wenn er in Quartus II bereits kompiliert wurde.

4.5 Zuweisungen (Assignments)

Im Menü **Assignments** können verschiedenste Zuweisungen erfolgen. Dort gibt es Untermenüeinträge, über die die Ein- und Ausgänge des Entwurfs den Anschlüssen eines Bauteils zugeordnet (**Assignments–Assign Pins**) oder einzelne Schaltungsteile in bestimmte Bereiche eines Bausteins gelegt werden können.

4.6 Compiler und Simulator

Im **Processing** Menü sind alle Kommandos zusammengefasst, die für das Compilieren und die Simulation des Schaltungsentwurfs notwendig sind. In den meisten Fällen beschränkt sich das Prozedere auf den Start des Compilers und des Simulators über die allgemeine Symbolleiste. Nach dem Durchlauf öffnet sich automatisch ein Ergebnisfenster (Compilation Report oder Simulation Report) in dem die Ergebnisse betrachtet werden können.

4.6.1 Ergebnisfenster (Reports)

In den **Reports** werden die Ergebnisse der Compilierung oder der Simulation aufgelistet. Dabei werden nicht nur Fehlermeldungen ausgegeben, sondern auch eine Zusammenfassung, die Einstellungen, die vom Compiler generierten Gleichungen, die **Timing–Analysen** und mehr. Bei der Compilierung ist der Floorplan von besonderem Interesse. In ihm werden die realisierten Verbindungen und Belegungen des Bausteins angezeigt. Weiterhin kann man einzelne Verbindungen und die jeweils auf ihnen entstehenden Laufzeiten betrachten. Bei der Simulation wird im Report–Fenster das Ergebnis in einer der **Vector Waveform Datei** äquivalenten Form angezeigt. Aus den Report–Fenstern können die Ergebnisse dann auch ausgedruckt werden.

4.7 Programmierung

Das im Praktikum verwendete DSP–Kit wird mit einem so genannten Byte-Blaster Kabel per USB an den PC angeschlossen. Gestartet wird die Programmierung unter dem Menü **Tools–Programmer** oder über die allgemeine Symbolleiste. Im sich öffnenden Fenster wird links oben die verwendete Hardware eingestellt (Byte Blaster [USB0]). Falls die erforderliche Datei (Endung *.sof „SRAM Object File“) nicht angezeigt wird, muss sie hinzugefügt werden. Danach kann der Programmiervorgang gestartet werden.

5 Die Entwicklungsumgebung im Praktikum

In diesem Kapitel wird die Entwicklungsumgebung beschrieben, die Ihnen im Praktikum zur Verfügung steht. Zuerst wird auf die Hardware, also das Entwicklungsboard mit seinen Komponenten eingegangen.

5.1 Das Altera DSP Entwicklungs–Board

Das Development Board stellt dem Entwickler folgende Ein- bzw. Ausgabemöglichkeiten zur Verfügung:

- Analog I/O
  - Zwei 14-bit analog-to-digital (A/D) Wandler mit 125 MSPS und 70 dB SNR
  - Zwei 14-bit digital-to-analog (D/A) Wandler mit 165 MSPS und 70 dB SNR
  - Ein 24-bit RGB VGA Adapter mit einem DB-15 Stecker
  - Ein Audio CODEC mit Eingang, Ausgang, und verstärktem Ausgang
- Memory Subsystem
  - 256 Mbyte DDR2 SDRAM DIMM
  - 1 Mbyte synchronous SRAM (SSRAM)
- Zwei EPCS64 64 Mbit seriell Konfigurationsbausteine
- Debugging Interface-Mictor Stecker zum Hardware und Software debugging
- Erweiterungsschnittstellen
  - 3.3-V/5-V tolerant Altera expansion/prototype headers
  - Ein Texas Instruments Evaluation Module (TI-EVM) Erweiterungsstecker
- Zwei sieben-segment LED Anzeigen
- Acht benutzerdefinierte LEDs
- Ein benutzerprogrammierbarer DIP Schalter (8 Positionen)
- Vier benutzerprogrammierbare Taster

Das zentrale Element bildet der programmierbare Logikbaustein EP2C70F672 von Altera. Eine Übersicht des gesamten Boards zeigt Bild 5.1. Die für das Praktikum wichtigsten Komponenten sollen im Folgenden kurz vorgestellt werden.

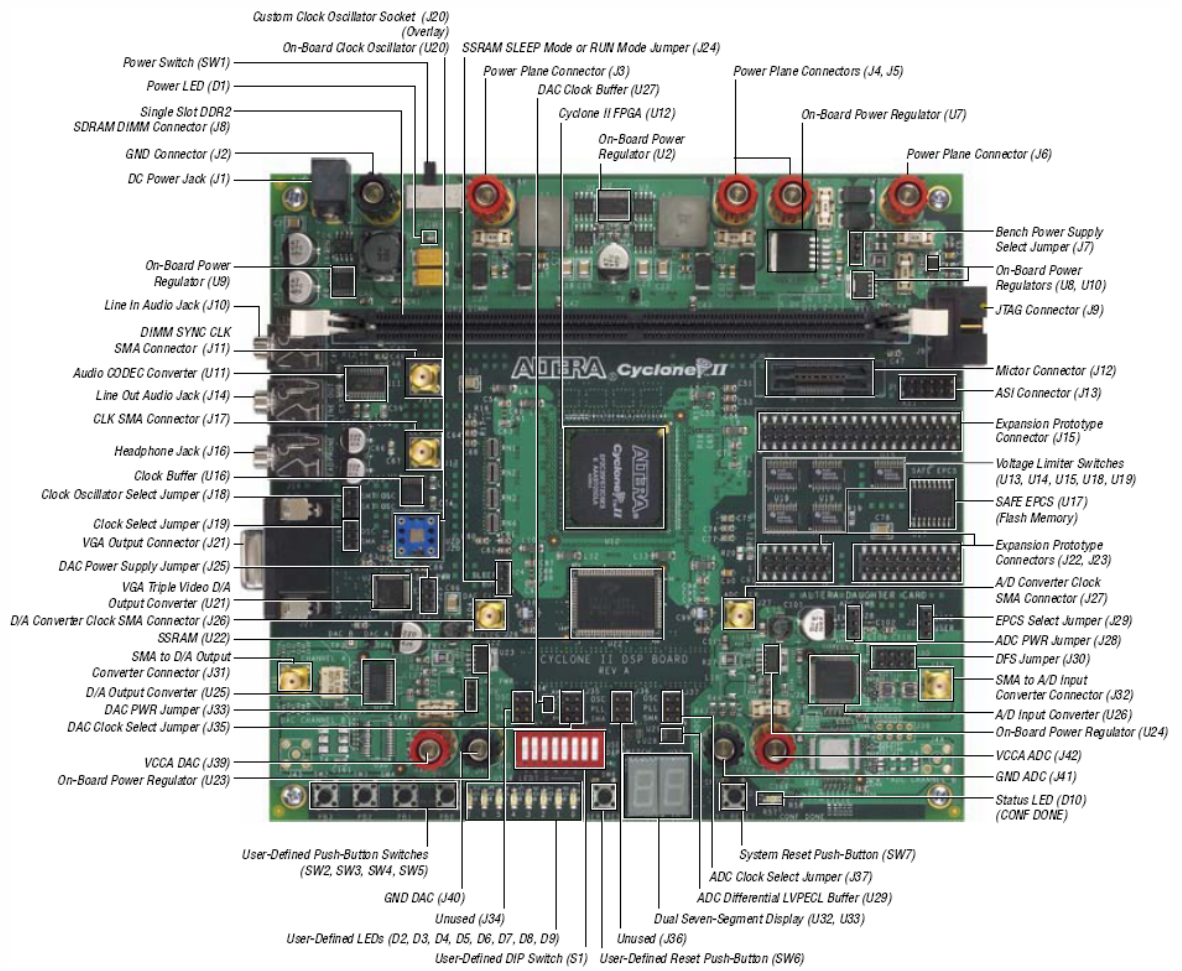

Bild 5.1: Übersichtsdarstellung des Altera Boards.

5.1.1 Der Altera EP2C70F672 Baustein

Aufgrund seiner Komplexität wird der Baustein den Field Programmable Gate Array Devices (FPGA) zugeordnet. Er besitzt ein 672–pin FineLine–BGA–Gehäuse (FBGA), wobei fast jeder zweite Pin entweder für Masse oder eine der beiden Versorgungsspannungen reserviert ist. Es stehen noch 422 benutzerdefinierte I/O–Ports zur Verfügung. Die Spannungen sind zum Einen die interne Versorgungsspannung von 1,8 V und zum Anderen die Spannung der I/O–Pins, die 3,3 V beträgt. Die logischen Eigenschaften des Bausteins sind in Tabelle 5.1 aufgelistet. Auf die einzelnen Elemente wird in den folgenden Unterkapiteln näher eingegangen.

Embedded 18×18 multipliers	150
Logic Elements (LEs)	68416
M4K RAM blocks (4 Kbits + 512 parity bits)	250
Maximum differential channels	262
PLLs	4 PLLs
Total RAM Bits	1152000
User I/O Pins	422

Tabelle 5.1: Eigenschaften des CPLD Bausteins.

5.1.1.1 Das Logic Element (LE)

Die kleinste Einheit des Bausteins bildet das Logic Element. Es besteht aus einer 4–Bit Look Up Table (LUT), in welcher die logischen Verknüpfungen abgelegt werden. Desweiteren ist ein programmierbares Register vorhanden, welches wahlweise als D–, T–, JK– oder RS–Flipflop eingesetzt werden kann. Als Ausgang steht das Ergebnis der LUT direkt und/oder im Ausgangsregister zwischengespeichert zur Verfügung. Bild 5.2 zeigt den vereinfachten Aufbau eines LEs. Für schnelle arithmetische Operationen gibt es eine spezielle Carry-Chain und für breite Funktionen eine Cascade–Chain, die die einzelnen LEs verbindet. Das LE kann in drei verschiedenen Modi programmiert werden. Dies sind der Counter Mode, der Arithmetic Mode und der Normal Mode, welche jeweils spezielle Vorteile bei bestimmten Schaltungen besitzen. Der Compiler übernimmt die Modi–Einstellung vollautomatisch, wenn nicht vom Benutzer eine bestimmte Vorgabe besteht.

5.1.1.2 Der Logic Array Block (LAB)

Jeweils zehn LEs bilden einen Logic Array Block. Innerhalb dieses Blockes gibt es schnelle lokale Verbindungen. Ein LAB besitzt je zwei clock und clock enable Signale. Alle LEs innerhalb des LABs können nur auf diese Signale zurückgreifen. Die clock enable–Signale gelten für den gesamten Block. Wird insbesondere von einem Taktsignal sowohl die steigende als auch die negative Flanke benötigt, so werden hierfür schon beide

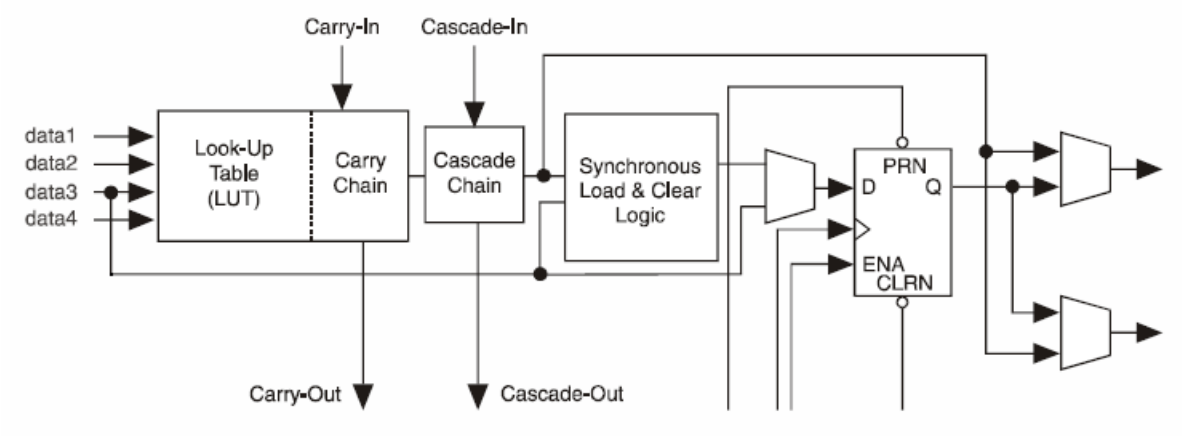

Bild 5.2: Vereinfachter Aufbau eines Logic Elements.

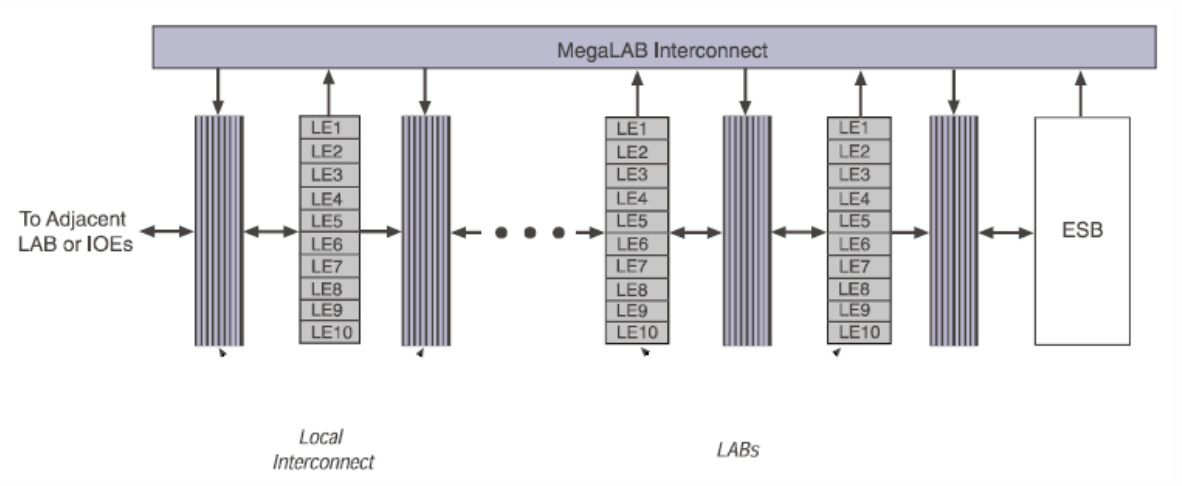

Bild 5.3: Die MegaLAB–Struktur.

clock–Leitungen des LABs belegt. Benötigt ein LE einen anderen Takt, so muss es in einem anderen LAB platziert werden.

5.1.1.3 Die MegaLAB–Struktur

In einer MegaLAB–Struktur sind 16 LABs und ein ESB gruppiert. Bild 5.3 veranschaulicht diese Hierarchie. Die lokalen Verbindungen und die sogenannten MegaLAB Verbindungen sind in diesem Schaubild ebenfalls dargestellt.

5.1.1.4 Der Embedded System Block (ESB)

Eine Besonderheit der Cyclone II Architektur stellt der Embedded System Block dar. Mit ihm kann entweder ein Set von Makrozellen oder Speicher realisiert werden. Dieser variable Einsatzbereich erhöht die Flexibilität des gesamten Bausteins. Hinsichtlich der

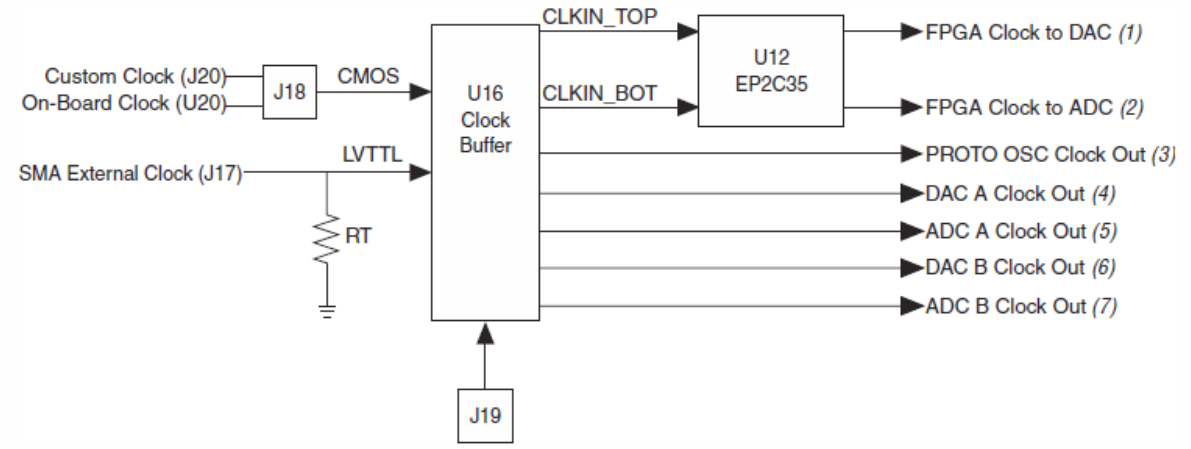

Bild 5.4: Taktverteilung des Altera Entwicklungs–Boards.

Clock- und Steuersignale gelten dieselben Einschränkungen wie bei den LEs innerhalb eines LABs (siehe Kap. 5.1.1.2). Wird der ESB als Speicher konfiguriert, so stehen eine Vielzahl von Organisationstypen (ROM, dual-port RAM, FIFO, CAM) zur Auswahl. Auch die Adressierung der 2048 Bits pro ESB ist variabel und reicht von 2048×1 bis 128×16. Breitere oder tiefere Speicher werden durch Kombination mehrerer Blöcke ermöglicht. Die Betriebsart dual-port RAM erlaubt gleichzeitiges schreiben und lesen des Speichers von verschiedenen Adressen. Der Programmierer muss allerdings sicherstellen, dass nicht simultan auf dieselbe Adresse zugegriffen wird, sonst ist das Ergebnis nicht vorhersagbar.

5.1.1.5 Globale Signale

Es gibt acht besonders ausgelegte globale Verbindungen auf dem Chip, wobei vier davon für Taktsignale reserviert sind (siehe Kap. 5.1.2). Die übrigen vier stehen dem Entwickler zur freien Verfügung. Sie können entweder von einem externen Signal oder von interner Logik betrieben werden. Findet der Quartus–Compiler ein asynchrones Reset- oder Setsignal, so legt er dieses automatisch auf eine der globalen Leitungen, falls eine davon verfügbar ist.

5.1.2 Taktung und PLLs

Die Taktverteilung des Altera Entwicklungs–Boards ist in Bild 5.4 dargestellt. Zunächst muss der Takt definiert werden. Dies geschieht mit Hilfe des clock–buffers U16. Dieser generiert acht identische clock–Ausgänge (davon ist ein Ausgang nicht verbunden und zwei Ausgänge nicht in Verwendung), die auf dem Board zur Verfügung stehen. Der Takt kann dabei aus den folgenden Optionen gewählt werden:

- Verwendung des on–board Taktoszillator mit 100 MHz (U20)
- Verwendung eines eigenen Taktoszillators (Ersatz des 100 MHz Oszillators in U20)

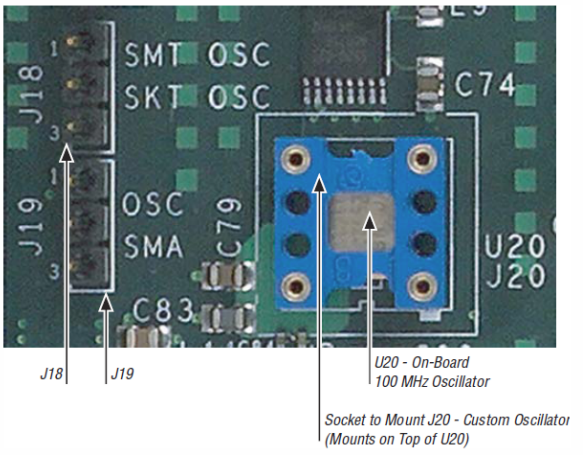

Bild 5.5: U20/J20, J18 & J19 für die Einstellung des Taktes.

- Verwendung eines externen Taktes über die SMA Stecker (J20)

In Bild 5.5 ist der on–board Oszillator und die Jumper zur Einstellung dargestellt. Für die Versuche im Rahmen dieses Praktikums ist der on–board Oszillator völlig ausreichend. Für einige Versuchsteile muss er sogar zusätzlich intern verringert (geteilt) werden, um z.B. die Verschiebung im Lauflichtversuch für das menschliche Auge sichtbar zu machen. Die Zuweisung des Global Clock innerhalb der Praktikumsversuche geschieht über die Pinnumer N25.

Cyclone® II Bausteine besitzen bis zu vier phase-locked Schleifen (PLLs), die robustes Taktgebermanagement bzw. -synthese für Vorrichtungstaktgebermanagement, Taktgebermanagement des externen Systems und Input-/Outputschnittstellen zur Verfügung stellen. Cyclone II PLLs sind vielseitig einsetzbar und können als Nullverzögerungspuffer, Jitterdämpfer oder Frequenzsynthesizer verwendet werden. Die bis zu vier PLLs unterstützen fortgeschrittene Funktionen, wie Taktgeberumschalten und programmierbares Umschalten. Diese PLLs bieten die Möglichkeit der Taktgebervervielfachung bzw. -reduzierung, Phasenverschiebung und programmierbarer Taktzyklen und können verwendet werden, um Taktgeberverzögerungen herabzusetzen bzw. zu justieren. Die Cyclone II Familie unterstützt zudem auch einen Energiesparmodi, in dem unbenutzte Taktgebernetze abgestellt werden können. Die Altera Quartus II Software bietet Zugriff auf die PLLs und ihren Eigenschaften ohne irgendwelche externen Vorrichtungen zu erfordern. Die bis zu vier PLLs sind in den vier Ecken des Bausteins angeordnet. Der Hauptzweck einer PLL ist es, die Phase und die Frequenz des VCO zu einem Bezugstaktgeber zu synchronisieren. Es gibt einige Bestandteile, die eine PLL enthalten, um diese Phasenausrichtung zu erzielen. Ausführliche Details zur Verwendung der PLLs der Cyclone II Familie findet sich unter `http://www.altera.com/literature/hb/cyc2/cyc2_cii51007.pdf`.

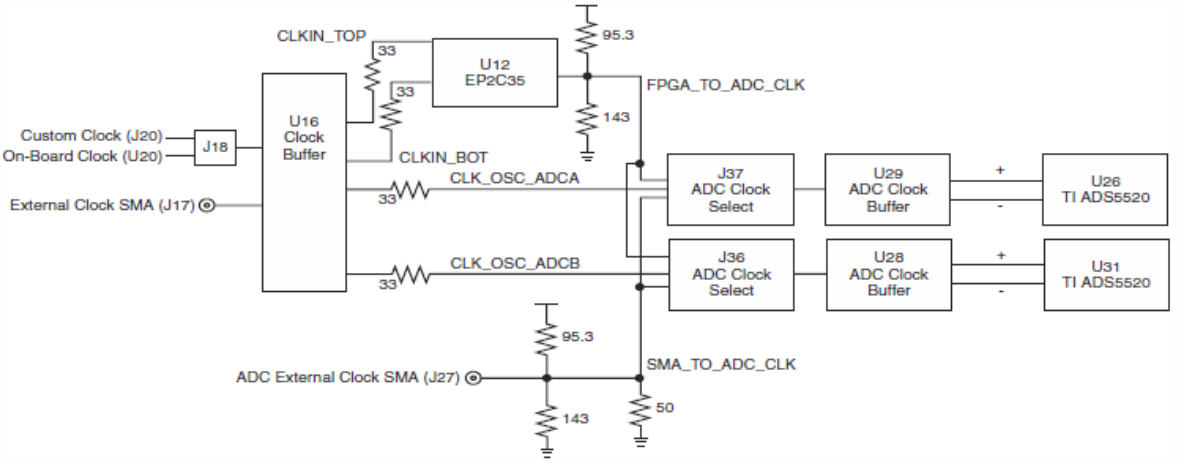

Bild 5.6: Schematische Darstellung der Taktoptionen der A/D–Wandler.

5.1.3 A/D–Wandler

Es sind zwei 14 Bit A/D–Wandler vom Typ TI ADS5500 auf dem Entwicklungs–Board vorhanden. Das Signal–to–noise–ratio (kurz: SNR) beträgt 70 dB für Eingangssignale von 1 MHz bis zur Nyquistfrequenz der Wandler. Die maximale Eingangsspannung der Wandlereingänge beträgt $2,2V_{PP}$. Das Datenformat des gewandelten Analogsignals kann durch Jumper (J30 für Kanal A und J38 für Kanal B) auf dem Board eingestellt werden. Tabelle 5.2 gibt die möglichen Datenformate wieder (für den Versuch mit digitalen Filtern kann hier das standardmäßig eingestellte Zweierkomplement–Datenformat verwendet werden!). Bild 5.6 zeigt die Komponenten, die für die Taktauswahl für den A/D–Wandler TI ADS5500 (U26 für Kanal A, U31 für Kanal B) verwendet werden. Die Jumper J37 (Kanal A) oder J36 (Kanal B) wählen das Taktsignal aus folgenden Möglichkeiten:

- on–board Oszillator,
- FPGA Takt oder
- externer SMA–Takt.

Das gewählte Taktsignal wird über einen differentiellen LVPECL Buffer (U29 für Kanal A, U28 für Kanal B) an den Wandlereingang geführt. Eine schematische Darstellung eines Wandelvorganges ist in Bild 5.7 in Form des Timingdiagramms dargestellt. Zur Verwendung der A/D–Wandler in den Versuchen sind die Pinzuweisungen nach Tabelle 5.3 für Kanal A und Tabelle 5.4 für Kanal B notwendig. Beachten Sie dabei die zusätzlichen Pinnummern, die manuell auf GND und V_{CC} zu legen sind!

5.1.4 D/A–Wandler

Auf dem Entwicklungs–Board sind zwei 14 Bit D/A–Wandler vom Typ TI DAC904E

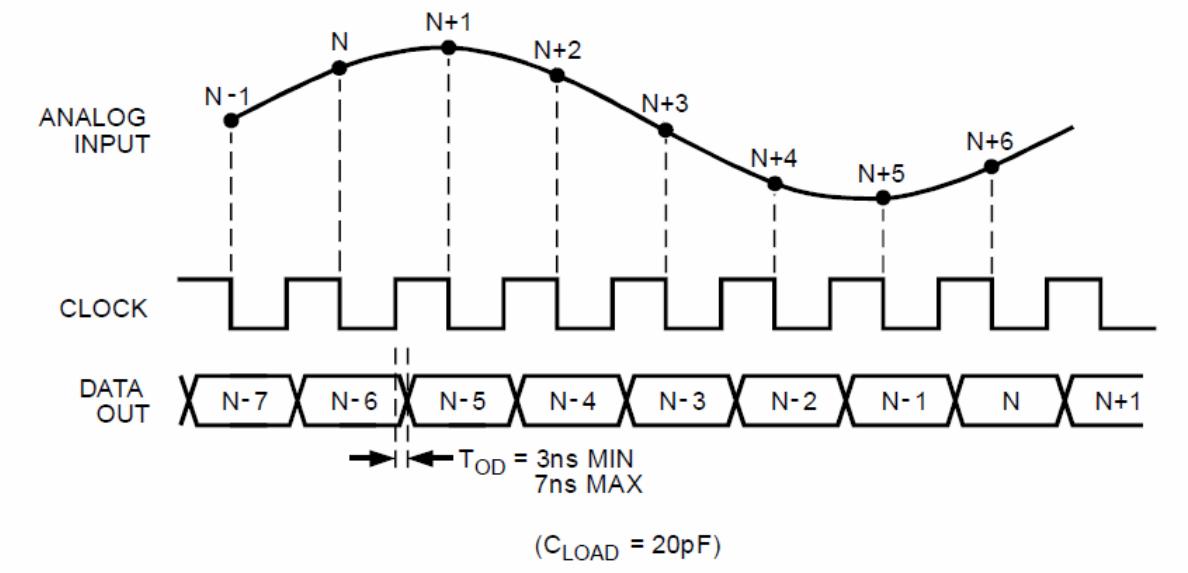

Bild 5.7: Timingdiagramm A/D–Wandler.

Jumper (J30 & J38) Einstellung	**Data Format**	**Clock output Polarität**
Pins 1 und 2	Zweierkomplement	Daten gültig bei fallender Flanke
Pins 3 und 4	Binär	Daten gültig bei fallender Flanke
Pins 5 und 6	Zweierkomplement	Daten gültig bei steigender Flanke
Pins 7 und 8	Binär	Daten gültig bei steigender Flanke

Tabelle 5.2: Einstellbare Datenformate der A/D–Wandler.

A/D–Wandler Pinname	A/D–Wandler Pinnummer	Cyclone II Pinnummer		A/D–Wandler Pinname	A/D–Wandler Pinnummer	Cyclone II Pinnummer
ADC_A_CLK_N	11			ADC_A_D5	52	C11
ADC_A_CLK_P	10			ADC_A_D6	53	B12
ADC_A_CM	17			ADC_A_D7	54	D13
ADC_A_DCLK	43	A13		ADC_A_D8	55	B22
ADC_A_DFS	40			ADC_A_D9	56	A21
ADC_A_INM	20			ADC_A_D10	60	A23
ADC_A_INP	19			ADC_A_D11	61	B23
ADC_A_IREF	31			ADC_A_D12	62	C22
ADC_A_OE	41	F7 → VCC		ADC_A_D13	63	A22 (MSB)
ADC_A_OVR	64	D15		ADC_A_REFM	30	
ADC_A_D0	44	C5 (LSB)		ADC_A_REFP	29	
ADC_A_D1	45	C6		ADC_A_SEN	4	B18
ADC_A_D2	46	B7		ADC_RESET	35	T24 → GND
ADC_A_D3	47	A8		ADC_SCLK	2	AD24
ADC_A_D4	51	A9		ADC_SDATA	3	Y1

Tabelle 5.3: A/D ADS5500 A/D–Wandler Pinnummern für Kanal A.

A/D–Wandler Pinname	A/D–Wandler Pinnummer	Cyclone II Pinnummer		A/D–Wandler Pinname	A/D–Wandler Pinnummer	Cyclone II Pinnummer
ADC_B_CLK_N	11			ADC_B_D5	52	B20
ADC_B_CLK_P	10			ADC_B_D6	53	A20
ADC_B_CM	17			ADC_B_D7	54	B21
ADC_B_DCLK	43	C13		ADC_B_D8	55	F18
ADC_B_DFS	40			ADC_B_D9	56	G18
ADC_B_INM	20			ADC_B_D10	60	E18
ADC_B_INP	19			ADC_B_D11	61	F20
ADC_B_IREF	31			ADC_B_D12	62	D21
ADC_B_OE	41	R2 → VCC		ADC_B_D13	63	D20 (MSB)
ADC_B_OVR	64	A6		ADC_B_REFM	30	
ADC_B_D0	44	F17 (LSB)		ADC_B_REFP	29	
ADC_B_D1	45	D17		ADC_B_SEN	4	D19
ADC_B_D2	46	D18		ADC_RESET	35	T24 → GND
ADC_B_D3	47	C19		ADC_SCLK	2	AD24
ADC_B_D4	51	B19		ADC_SDATA	3	Y1

Tabelle 5.4: A/D ADS5500 A/D–Wandler Pinnummern für Kanal B.

integriert. Den zeitlichen Ablauf einer Wandlung zeigt exemplarisch Bild 5.8. In Tabelle 5.5 und Tabelle 5.6 sind die dazugehörigen Parameter angegeben. Der D/A–Wandler arbeitet mit der positiven Taktflanke und das analoge Ausgangssignal liegt um einen Takt und t_{pd} verzögert am Ausgang an. Zu beachten ist hierbei, dass sich die Verzögerung auf einen Takt des D/A–Wandlerbausteins bezieht, der unabhängig vom A/D–Wandlertakt sein kann. Die Signalverzögerung summiert sich somit zu einem D/A–Wandlertakt plus 1 ns. In den in dieser Arbeit behandelten Projekten kann diese Verzögerung allerdings vernachlässigt werden, da sich eine Messung der Verzögerungszeiten der Filter nur bei einem externen Takt von maximal 500 kHz anbietet.

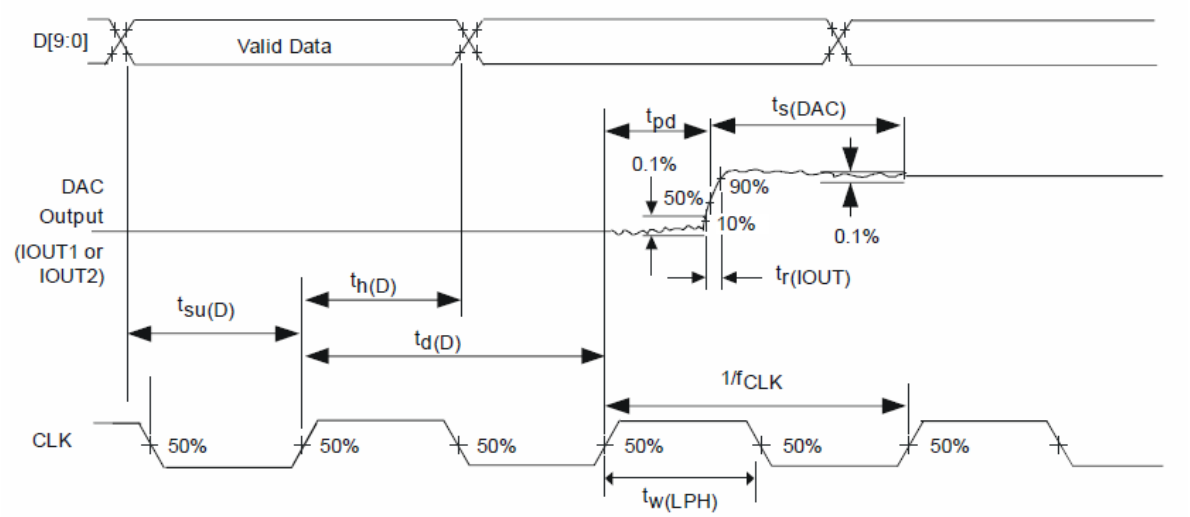

Bild 5.8: Exemplarischer Timingdiagramm einer D/A–Wandlung.

Der D/A–Wandler wird bei den Praktikumsprojekten immer mit 100 MHz betrieben (siehe Kap. 5.1.2), somit fällt seine Verzögerungszeit gegenüber einer Abtastperiodendauer nicht ins Gewicht. Die D/A–Wandler werden über die SMA–Stecker mit dem Interface des TI DAC904E Wandlerbausteins mit Kabeln verbunden. Die D/A–Wandlerbausteine liefern ein 14 Bit aufgelöstes Signal mit einer Datenrate bis zu 165 MSPS. Das Signal–to–noise–ratio (kurz: SNR) beträgt 70 dB für Ausgangssignale von 1 MHz bis zur Nyquistfrequenz der Wandler. Die beiden Wandler sind den logischen Kanälen A und B zugeordnet. Zur Verwendung müssen die in Tabelle 5.7 bzw. Tabelle 5.8 angegebenen Pinzuweisung durchgeführt werden. Das Datenformat der D/A–Wandlerbausteine ist Einerkomplement!

5.1.5 Speicherbausteine

Auf dem Board befinden sich noch zwei RAM–Bausteine von IDT mit je 64 K × 16 Bit. Sie besitzen eine Zugriffs- und Zykluszeit von 12 ns. Die korrekte Ansteuerung muss der Programmierer sicherstellen. Im Rahmen dieses Praktikums wird der externe Speicher nicht direkt verwendet.

Parameter	Test Conditions	Min.	Typ.	Max.	Unit
Analog output					
f_{CLK} Maximum output update rate	DV_{DD} = 4,5 V to 5,5 V	100	125		MSPS
	DV_{DD} = 4,5 V to 5,5 V	70	100		
t_{sDAC} Output settling time to 0,1 %			35		ns
t_{pd} Output propagation delay			1		ns
GE Glitch energy	Worst case LSB transition (code 511 – 512)		5		pV-s
$t_{r(IOUT)}$ Output rise time 10 to 90 %			1		ns
$t_{f(IOUT}$ Output fall time 90 to 10 %			1		ns
Output noise	$IOUT_{FS}$ = 20 mA		15		$\frac{pA}{Hz}$
	$IOUT_{FS}$ = 2 mA		10		

Tabelle 5.5: AC Spezifikationen des D/A–Wandlers.

Parameter	Test Conditions	Min.	Typ.	Max.	Unit
Timing					
$t_{su(D)}$ Input setup time		1			ns
$t_{h(D)}$ Input hold time		1			ns
$t_{w(LPH)}$ Input latch pulse high time		4			ns
$t_{d(D)}$ Digital delay time				1	clk

Tabelle 5.6: Digitale Spezifikationen des D/A–Wandlers.

D/A–Wandler (U25) Pinname	D/A–Wandler (U25) Pinnummer	Cyclone II (U12) Pinnummer
DAC_A_D0	14	AB1 (LSB)
DAC_A_D1	13	AA1
DAC_A_D2	12	AE3
DAC_A_D3	11	AD3
DAC_A_D4	10	U3
DAC_A_D5	9	T2
DAC_A_D6	8	Y4
DAC_A_D7	7	AA5
DAC_A_D8	6	V5
DAC_A_D9	5	V6
DAC_A_D10	4	P3
DAC_A_D11	3	U7
DAC_A_D12	2	R5
DAC_A_D13	1	P6 (MSB)
DAC_A_IOUTn	21	
DAC_A_IOUTp	22	

Tabelle 5.7: TI DAC904E D/A–Wandler Pinnummern für Kanal A.

D/A–Wandler (U25) Pinname	D/A–Wandler (U25) Pinnummer	Cyclone II (U12) Pinnummer
DAC_B_D0	14	M4 (LSB)
DAC_B_D1	13	M5
DAC_B_D2	12	U20
DAC_B_D3	11	V20
DAC_B_D4	10	V21
DAC_B_D5	9	B24
DAC_B_D6	8	T23
DAC_B_D7	7	P23
DAC_B_D8	6	Y24
DAC_B_D9	5	V24
DAC_B_D10	4	W25
DAC_B_D11	3	W26
DAC_B_D12	2	V25
DAC_B_D13	1	T25 (MSB)
DAC_B_IOUTn	21	
DAC_B_IOUTp	22	

Tabelle 5.8: TI DAC904E D/A–Wandler Pinnummern für Kanal B.

5.1.6 Schalter und Taster

Im Bereich der Spannungsversorgung befindet sich der Power Switch (SW1). Mit ihm wird das Board eingeschaltet, was durch eine SMD–LED mit blauem Aufleuchten quittiert wird. Weiterhin findet sich ein DIP–Schalter nach Bild 5.9 auf dem Board. Dieser kann z.B. für das Laden eines Musters im Praktikumsversuch „Lauflicht“ verwendet werden. Die Pinzuordnung der DIP–Schalter muss dabei nach Tabelle 5.9 vorgenommen werden. Neben dem DIP–Schaltern befinden sich weitere Taster auf dem Board.

DIP Switch Label	Board Referenz	Schematic Signalname	Cyclone II (U12) Pinnummer
1	0	USER_DIPSW0	AC13
2	1	USER_DIPSW1	A19
3	2	USER_DIPSW2	C21
4	3	USER_DIPSW3	C23
5	4	USER_DIPSW4	AF4
6	5	USER_DIPSW5	AC20
7	6	USER_DIPSW6	AE18
8	7	USER_DIPSW7	AE19

Tabelle 5.9: Benutzerdefinierte Schalter des DIP–Schalters.

Zu beachten ist, dass der Taster 7 (SW7) fest mit dem globalen Reset verdrahtet ist,

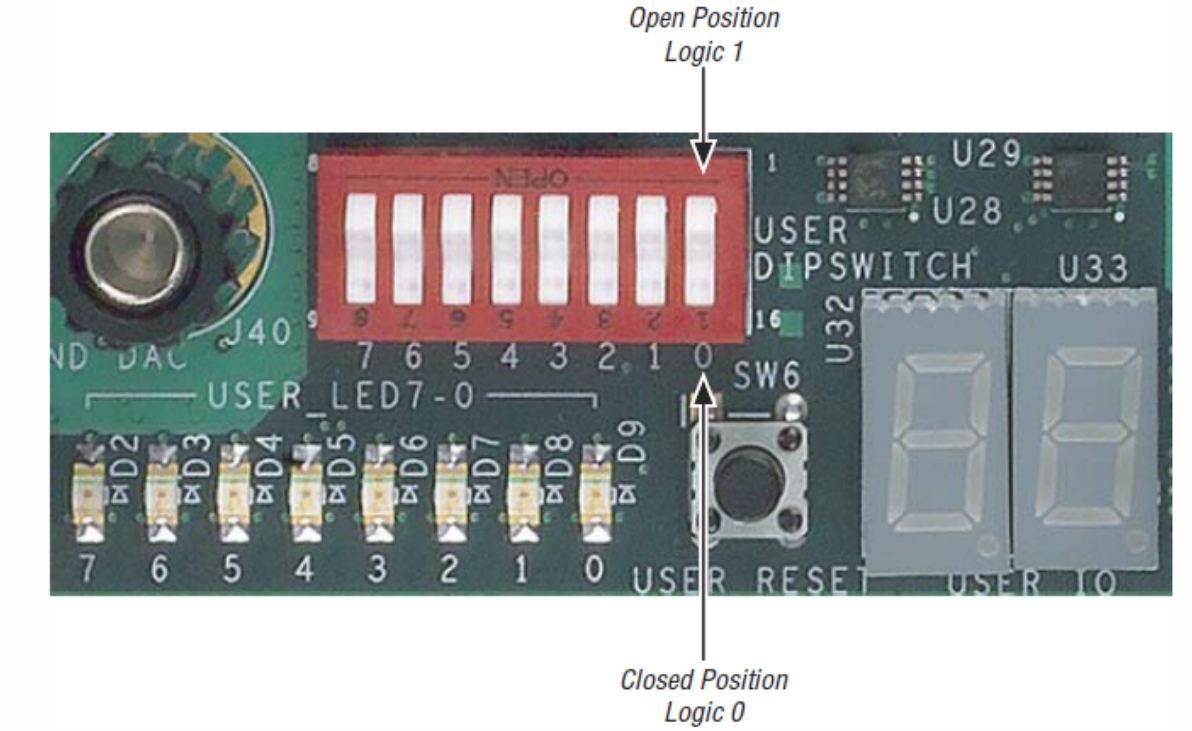

Bild 5.9: Benutzerdefinierbarer DIP–Schalter.

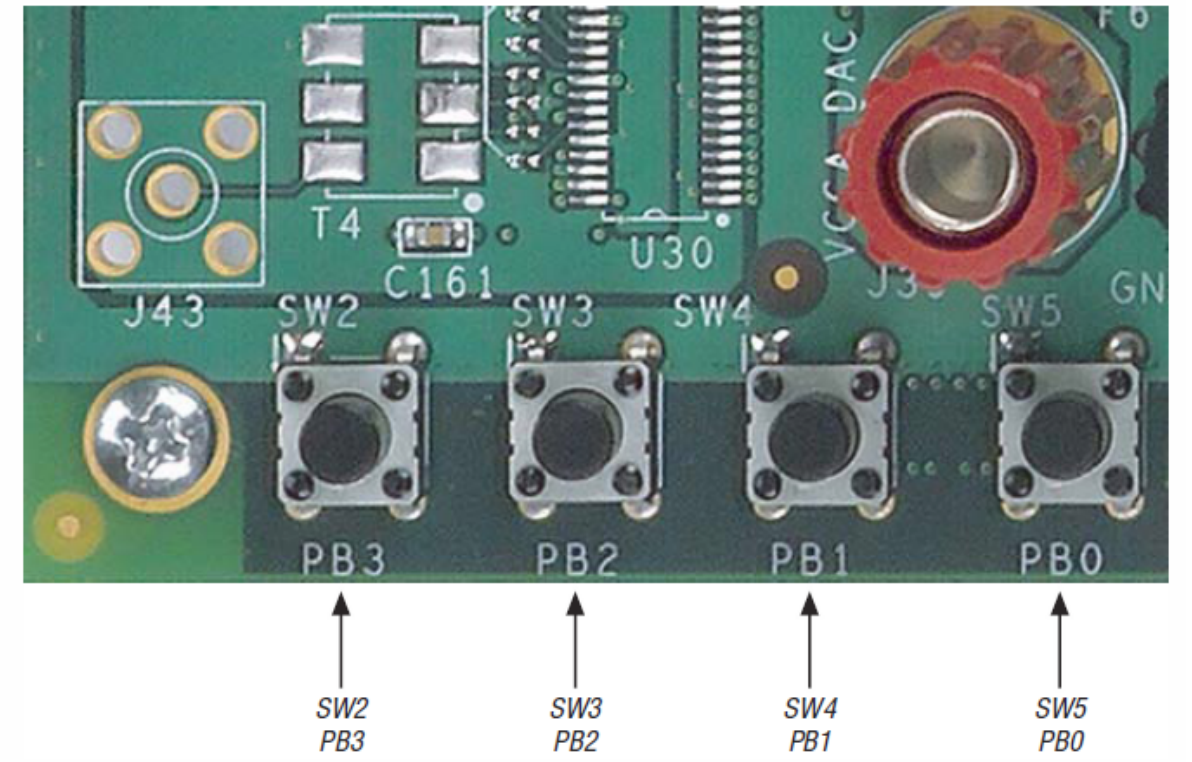

Bild 5.10: Benutzerdefinierte Taster.

das Entwicklungs–Board in den Grundzustand zurücksetzt und somit nicht für benutzerdefinierte Eingaben verwendet werden kann. Hinzu kommt der Taster SW6 für den Anwenderreset. Die übrigen Taster lassen sich aber beliebig verwenden und sind relativ gut zu bedienen. Eine Entprellung bei der Verwendung der Taster ist vorzunehmen und es ist zu beachten das alle Taster „active–low“ sind (ein „Entprellmodul“ ist beim Betreuer erhältlich).

Taster Name	**Board Referenz**	**Schematic Signalname**	**Pin Nummer**
PB3	SW2	USER_PB3	AE14
PB2	SW3	USER_PB2	AE22
PB1	SW4	USER_PB1	AE16
PB0	SW5	USER_PB0	AC18
	SW6 (user-defined)	USER_RESETn	A14

Tabelle 5.10: Pinzuweisung der Taster.

5.1.7 Leuchtdioden

Das Entwicklungsboard besitzt eine Reihe aus acht Leuchtdioden in SMD–Ausführung für die benutzerdefinierte Verwendung. Die Power–LED (D1) leuchtet blau auf, sobald eine Versorgungsspannung von 5 V am Board anliegt. Die Leuchtdiode mit der Bezeichnung Conf_Done (D10) leuchtet auf, sobald der FPGA–Baustein erfolgreich programmiert wurde und betriebsbereit ist. Die übrigen acht Leuchtdioden kann der Programmierer über den Cyclone II Baustein ansteuern. Dabei ist zu beachten, dass eine logische „0“ am Ausgang die Diode zum Leuchten bringt und sie bei einer logischen „1“ erlischt („low–active“). Die Platzierung der Leuchtdioden kann Bild 5.11 und die Pinnummern der Tabelle 5.11 entnommen werden.

LED Nummer	**Board Referenz**	**Schematic Signalname**	**Cyclone II (U12) Pinnummer**
7	D2	USER_LED7	AA7
6	D3	USER_LED6	AA6
5	D4	USER_LED5	AB4
4	D5	USER_LED4	AC3
3	D6	USER_LED3	E22
2	D7	USER_LED2	F20
1	D8	USER_LED1	B3
0	D9	USER_LED0	E5

Tabelle 5.11: Benutzerdefinierte LED Pins.

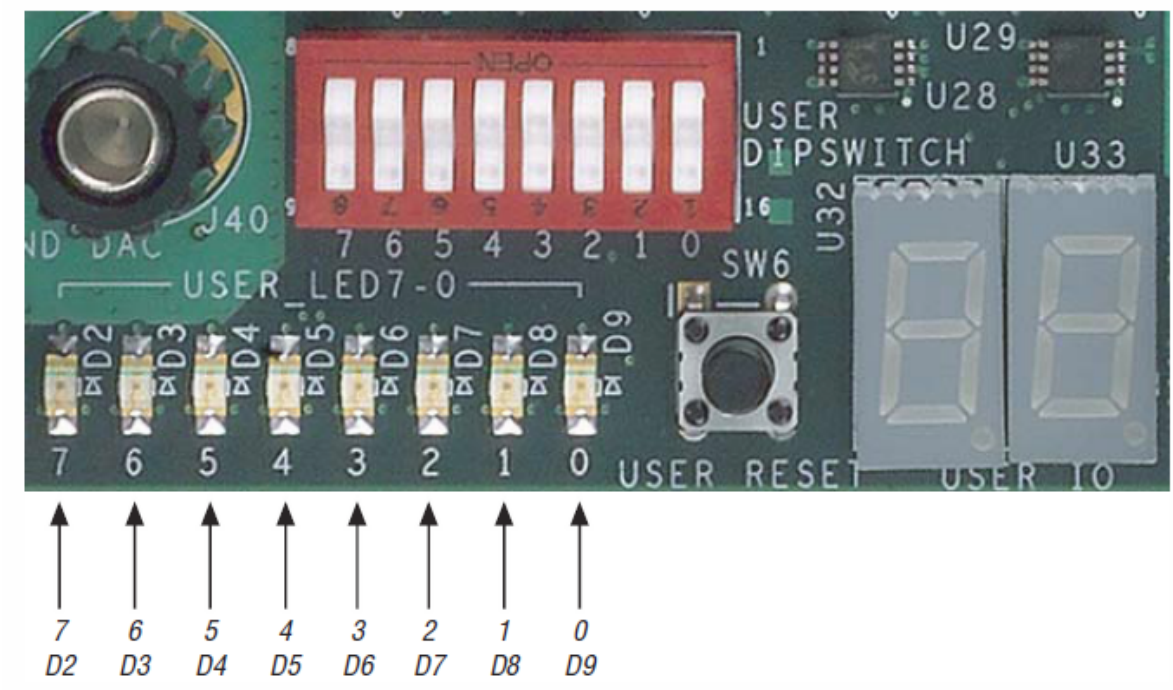

Bild 5.11: Platzierung der 8 benutzerdefinierbaren LEDs auf dem Altera Entwicklungs–Board.

5.1.8 Sieben–Segment–Anzeige

Weiterhin befindet sich eine Sieben–Segment–Anzeige (Bild 5.12) auf dem Altera Entwicklungs–Board, die vom Benutzer frei programmiert werden kann. Die Zuweisungen der Pins ist in Tabelle 5.12 angegeben. Auch die Sieben–Segment–Anzeige ist „active–low“.

U32			U33		
Segment	**Schematic Signalname**	**Pin Nummer**	**Segment**	**Schematic Signalname**	**Pin Nummer**
A	DIG_MSB_A	Y21	A	DIG_LSB_A	K2
B	DIG_MSB_B	T7	B	DIG_LSB_B	U25
C	DIG_MSB_C	AB23	C	DIG_LSB_C	AA3
D	DIG_MSB_D	Y5	D	DIG_LSB_D	V1
E	DIG_MSB_E	E1	E	DIG_LSB_E	V7
F	DIG_MSB_F	U1	F	DIG_LSB_F	U23
G	DIG_MSB_G	W21	G	DIG_LSB_G	AC2
DP	DIG_MSB_DP	V3	DP	DIG_LSB_DP	P7

Tabelle 5.12: Pinzuordnung der Sieben–Segment Anzeige.

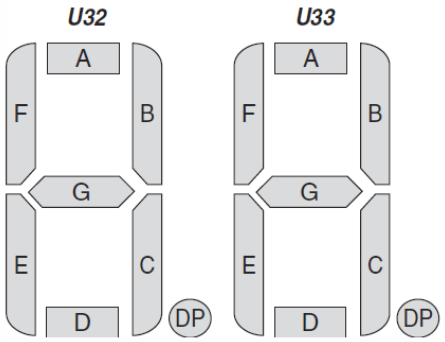

Bild 5.12: Zuordnung der einzelnen Segmente zur Tabelle 5.12.

6 Projekt Faltungscodierer

6.1 Theorie

6.1.1 Codierung

Wenn ein diskreter Kanal ohne Rauschen die Übertragung von N(T) möglichen Signalen der Zeitdauer T zulässt, ist nach Shannon die Kapazität C dieses Kanals durch

$$C = \lim_{T \to \infty} \frac{\log_2 N(T)}{T} \tag{6.1}$$

gegeben. Für einen Kanal mit Rauschen kann die Kapazität mit

$$C = MAX(H(X) - H_Y(X)) \tag{6.2}$$

berechnet werden. Die Ausdrücke $H(X)$ bzw. $H_Y(X)$ sind dabei keine Funktionen, sondern die Entropie bzw. bedingte Entropie der Zufallsvariablen X und Y. Dabei ist X die Zufallsvariable für die Signalquelle und Y die des Kanals. $H(X)$ stellt ein Maß für den Informationsgehalt von X dar, und $H_Y(X)$, die bedingte Entropie, die auch als $H(X|Y)$ geschrieben werden kann, lässt sich als Maß für die Unsicherheit oder Mehrdeutigkeit der empfangenen Symbole interpretieren. Die Maximierung erfolgt über alle möglichen Signalquellen. Allgemein lassen sich Informationsgehalte bzw. Unsicherheiten in der Form

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{6.3}$$

für die Wahrscheinlichkeiten p_1, ...,p_n , der n Realisierungen einer Zufallsvariablen, schreiben. Dabei gilt für die Summe der Wahrscheinlichkeiten:

$$H = -\sum_{i=1}^{n} p_i = 1. \tag{6.4}$$

Wird bei der Berechnung der in den Formeln angegebene Logarithmus zur Basis zwei verwendet, erhält man jeweils ein Ergebnis, das die Einheit Bit enthält. Wird z.B. der Logarithmus zur Basis zehn benutzt, enthält das Ergebnis als Einheit die Zahl der benötigten Dezimalstellen.

Beispiel:
Es sei eine binäre Symbolquelle gegeben, die eine Sequenz mit einer Rate von 1000 Symbolen pro Sekunde ausgibt, die Häufigkeit beider Symbole soll gleich sein. Damit ist eine

Zufallsvariable mit zwei Realisierungen und den Wahrscheinlichkeiten $p_1 = p_2 = 0,5$ gemeint. Diese Symbole sollen über einen Kanal mit einer Fehlerwahrscheinlichkeit von 5 % übertragen werden. Für die Quellentropie H(X) ergibt sich nun:

$$\begin{aligned} H(X) &= -(0,5 \cdot \log_2 0,5 + 0,5 \cdot \log_2 0,5) \\ &= 1 \text{ Bit/Symbol.} \end{aligned} \tag{6.5}$$

Die sich daraus ergebende Informationsbitrate ist:

$$\begin{aligned} R &= 1 \text{ Bit/Symbol} \times 1000 \text{ Symbol/s} \\ &= 1000 \text{ Bit/s} \end{aligned} \tag{6.6}$$

Dies ist auch das erwartete Ergebnis. Die Kanalentropie $H_Y(X)$ ist:

$$\begin{aligned} H_Y(X) &= -(0,95 \cdot \log_2 0,95 + 0,05 \cdot \log_2 0,05) \\ &= 0,286 \text{ Bit/Symbol.} \end{aligned} \tag{6.7}$$

Insgesamt resultiert daraus eine Kanalkapazität von:

$$\begin{aligned} C &= H(X) - H_Y(X) \\ &= 1 \text{ Bit/Symbol} - 0,286 \text{ Bit/Symbol} \\ &= 0,714 \text{ Bit/Symbol.} \end{aligned} \tag{6.8}$$

Bei der Übertragungsrate von 1000 Symbolen pro Sekunde resultiert dann eine mögliche Übertragungsrate von:

$$\begin{aligned} R &= 1000 \text{ Symbol/s} \times 0,714 \text{ Bit/Symbol} \\ &= 714 \text{ Bit/s.} \end{aligned} \tag{6.9}$$

Für die Übertragung der von der Quelle erzeugten Symbole ist es wichtig, dass diese möglichst mit der gleichen Häufigkeit auftreten, da sonst der Informationsgehalt einer Symbolsequenz sinkt. Dies ist in Bild 6.1 für eine Quelle mit zwei Symbolen gezeigt. Bei dieser Quelle ist $p_2 = 1 - p_1$, daher lässt sich der Informationsgehalt H allein über einer Variablen p auftragen. Shannon hat in seiner Arbeit bewiesen, dass es durch Codierung einer Quelle möglich ist deren Entropie zu maximieren und damit die Übertragungsrate bis zur maximal möglichen Rate, die durch den Kanal vorgegeben ist, zu steigern und dabei eine infinitesimal kleine Bitfehlerwahrscheinlichkeit zu erzielen. Aus diesem Beweis lassen sich jedoch keine Konstruktionsanweisungen für Codes ableiten, sondern er behandelt diese Codes nur allgemein. Der Vorteil codierter Übertragung kann wie folgt verdeutlicht werden:

Beispiel:
Die selbe Quelle wie im vorigen Beispiel soll nun codiert werden. Es sei ein Code mit der Rate $R = k/n = 1/2$ gegeben, d.h. für jedes binäre Symbol (mit einem Bit) der Quelle erzeugt ein Encoder ein 4–näres Symbol (mit zwei Bits). Die Symbolquelle und der Encoder werden dann als neue Quelle betrachtet, die eine Symbolrate von 1000 Symbolen

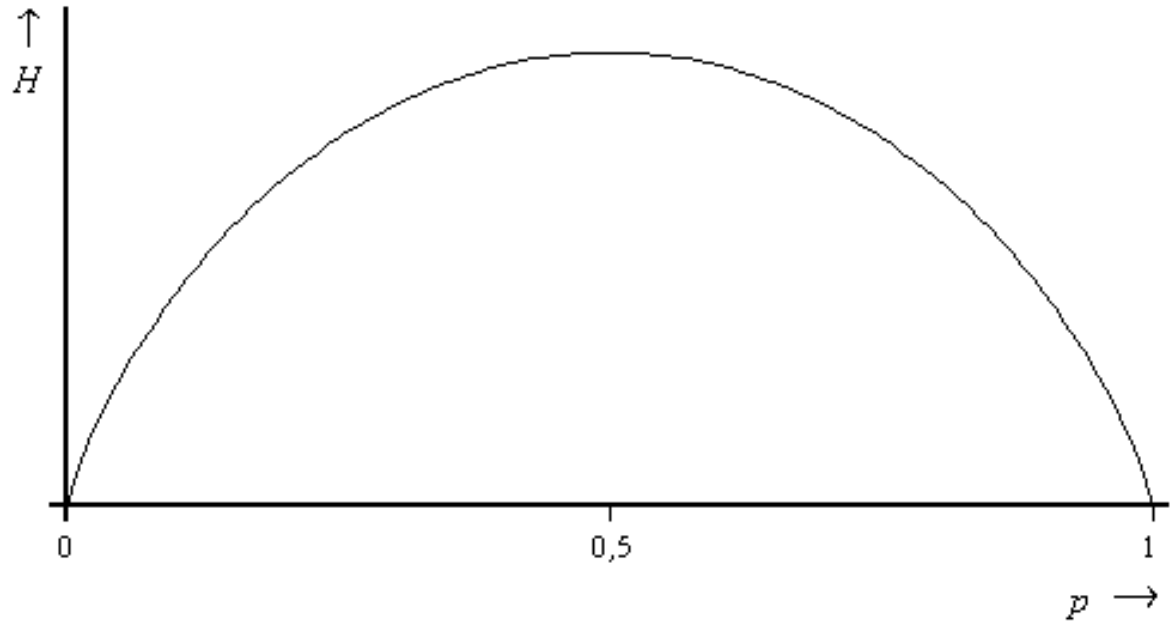

Bild 6.1: Entropie einer Quelle mit zwei Bit/Symbol.

pro Sekunde hat. Der Encoder soll die Symbole so erzeugen, dass sie unabhängig vom Eingang gleiche Häufigkeiten haben. Die Entropie der Quelle $H(X)$ ist dann:

$$\begin{aligned} H(X) &= -(4 \cdot 0,25 \cdot \log_2 0,25) \\ &= 2 \text{ Bit/Symbol.} \end{aligned} \tag{6.10}$$

Der Kanal soll die gleiche Wahrscheinlichkeit für einzelne Bitfehler haben wie im vorigen Beispiel und die einzelnen Fehlerereignisse sollen statistisch unabhängig voneinander sein. Damit ergeben sich die Symbolwahrscheinlichkeiten von 87,5 %, 5 %, 5 % und 0,25 %. Die Entropie des Kanals $H_Y(X)$ ist:

$$\begin{aligned} H_Y(X) &= -(0,875 \cdot \log_2 0,875 + 0,5 \cdot \log_2 0,5) \\ &= -0,5 \cdot \log_2 0,5 + 0,0025 \cdot \log_2 0,0025 \\ &= 1,19 \text{ Bit/Symbol.} \end{aligned} \tag{6.11}$$

Die daraus resultierende Kanalkapazität ist:

$$C = 2 \text{ Bit/Symbol} - 1,19 \text{ Bit/Symbol} = 0,81 \text{ Bit/Symbol.} \tag{6.12}$$

Mit der gegebenen Symbolrate resultiert eine mögliche Übertragungsrate von:

$$R = 1000 \text{ Symbol/s} \times 0,81 \text{ Bit/Symbol} = 810 \text{ Bit/s.} \tag{6.13}$$

Dies ist eine ca. 13 % höhere Bitrate im Vergleich zur uncodierten Übertragung. Für die Codierung und damit den Fehlerschutz von Daten gibt es verschiedene Arten von Codes. Die in modernen Kommunikationssystemen, wie z.B. den aktuellen und kommenden Mobilfunkstandards GSM und UMTS, verwendeten Codes sind vorallem so gennante Faltungscodes. Daneben gibt es noch die Gruppe von Blockcodes, hier soll nur auf die Faltungscodes eingegangen werden, die eine Untermenge der Baum–Codes sind.

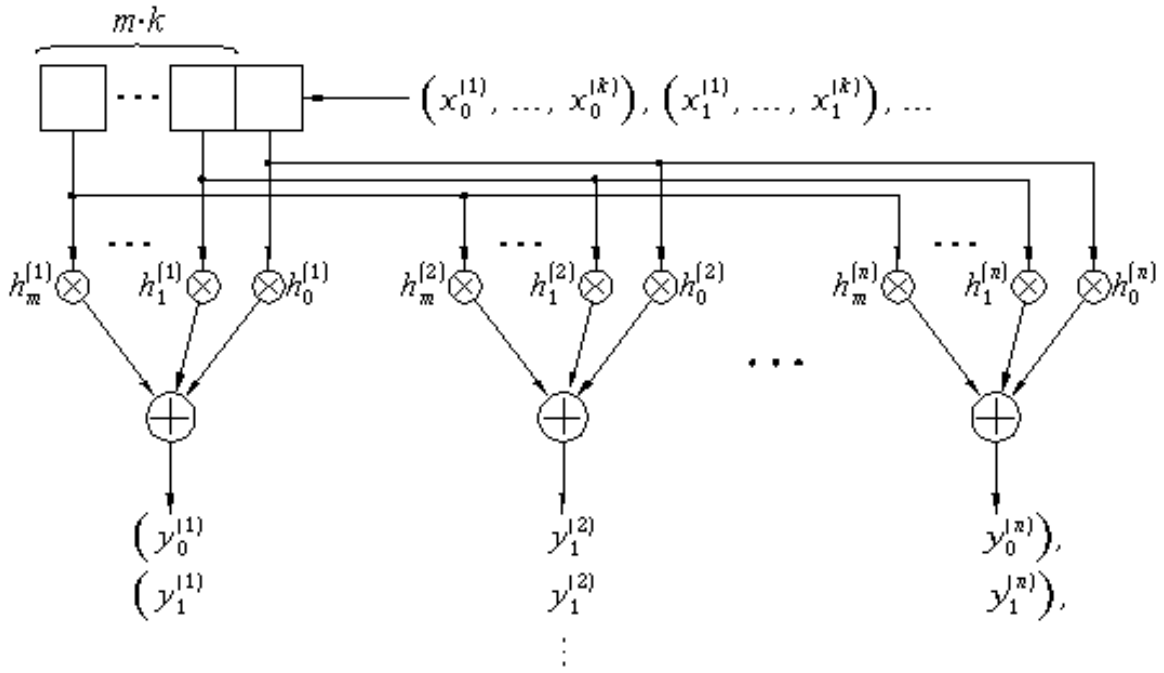

Bild 6.2: Allgemeiner Faltungscodierer.

6.1.1.1 Faltungscodierung

Im Gegensatz zu den Block–Codes hängen die Faltungscodes (convolutional codes) nicht nur von den letzten k sondern zusätzlich von den letzten $m \cdot k$ Informationssymbolen ab. Um die Eigenschaften dieser Codes vorherzusagen, ist man dann meist auf Simulationen angewiesen. Die Faltungscodes lassen sich entsprechend der Literatur in verschiedenen Formen darstellen, dabei liegt die mathematische Form den grafischen zugrunde. Der Vorteil der grafischen Darstellungen liegt jedoch in ihrer Anschaulichkeit, weswegen diese hier zuerst beschrieben werden.

6.1.1.2 Registerdarstellung

Eine einfach verständliche Darstellung ist dabei die als Schieberegister (Bild 6.2). Ohne Einschränkung der Allgemeinheit soll angenommen werden, dass die Informationssymbole binär sind und aus k Bit bestehen. Der Code wird dann durch ein Schieberegister der Länge $(m+1) \cdot k$ Bit und n linearen Verknüpfungen erzeugt. Pro Taktschritt wird ein Symbol (k Bit) in das Register geschoben und zusammen mit den übrigen $m \cdot k$ Bits des Registers zu n Codebits verknüpft. Die Coderate ist dann $R = k/n$. Die im Schieberegister gespeicherten $m \cdot k$ Bits werden auch als das Gedächtnis des Codierers bezeichnet, die Anzahl m ist die so genannte Gedächtnistiefe. Anhand eines speziellen Codierers mit $R = k/n = 1/2$ und $m = 2$ (Bild 6.3) sollen die weiteren grafischen Formen des Faltungscodes beschrieben werden.

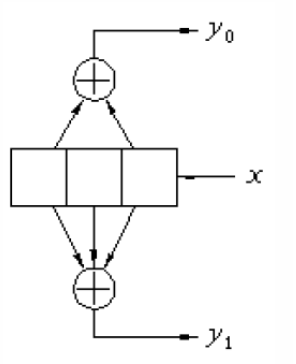

Bild 6.3: 1/2-Faltungscodierer mit m=2.

6.1.1.3 Codebaum

Werden die vom Codierer für alle möglichen Eingangssequenzen erzeugten Codefolgen von einem Knoten ausgehend aufgetragen, indem man für jedes mögliche Eingabesymbol von diesem Knoten einen Zweig zeichnet, daran den erzeugten Code notiert und so für die ganze Sequenz weiter verfährt, entsteht der so genannte Codebaum (Bild 6.4). In dieser Darstellung ist dann erkennbar, dass sich gewisse Strukturen rekursiv wiederholen. Für den in Bild 6.3 gezeigten Faltungscodierer ist dies nach einer Eindringtiefe von zwei der Fall. Dies korrespondiert mit der Gedächtnistiefe des Codierers. Die Eindringtiefe bezeichnet dabei die Anzahl von Eingabesymbolen, die notwendig ist, um vom Ursprungsknoten einen anderen Knoten zu erreichen. Da der Codebaum mit zunehmender Eindringtiefe exponentiell wächst, wird diese Form der Darstellung schnell unübersichtlich. Zur Beschreibung der sequenziellen Decodierung ist der Codebaum allerdings unerlässlich.

6.1.1.4 Netzdiagramm

Das Gedächtnis des Codierers kann $2^{(k \cdot m)}$ verschiedene Werte annehmen. Diese werden auch als Zustände bezeichnet. Stellt man einen Knoten für jeden Zustand dar und verknüpft diese mit Zweigen, wobei jeder der Zweige für einen zulässigen Übergang von einem Zustand zum darauf folgenden steht, erhält man das so genannte Netzdiagramm (trellis) (Bild 6.5). Da sich der Zustand i genau durch das Eingabesymbol vom Zustand $i+1$ unterscheidet, wird das sich ergebende Codesymbol eindeutig durch zwei aufeinander folgende Zustände festgelegt. Die Codesymbole werden wie im Codebaum wieder an den Zweigen notiert. Entsprechend der 2^k Eingabesymbole gehen von jedem Zustand 2^k Zweige aus. Im Diagramm in Bild 6.5 entspricht ein unterer Zweig, der von einem Knoten ausgeht, einem Eingabesymbol von 1, ein oberer einem Eingabesymbol von 0. Das Netzdiagramm ist zur Beschreibung des Viterbi–Decoders notwendig.

6.1.1.5 Zustandsdiagramm

Wenn die Zustände nur einmal als Knoten gezeichnet werden und alle möglichen Über-

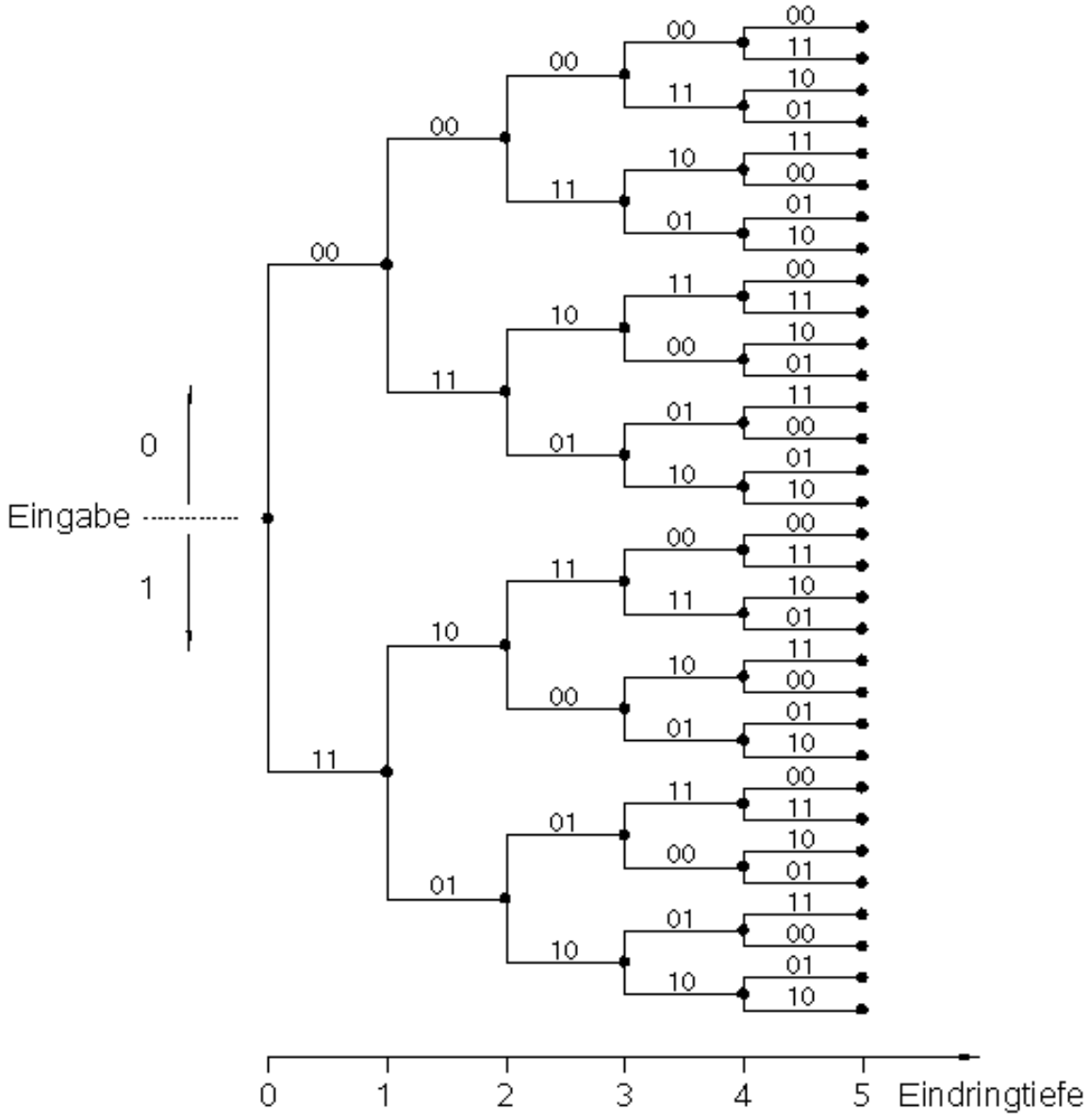

Bild 6.4: Codebaum eines 1/2–Faltungscodes mit $m = 2$.

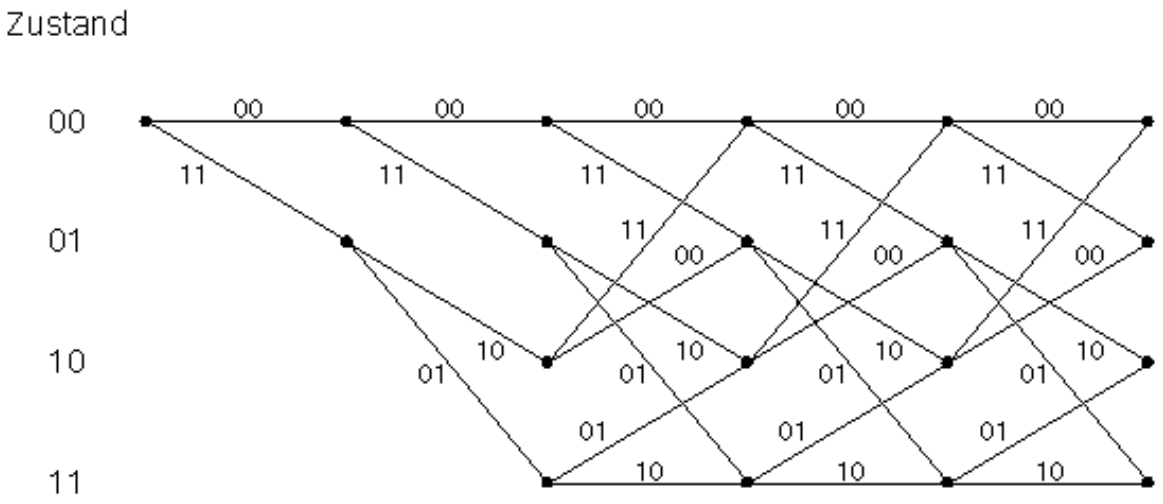

Bild 6.5: Netzdiagramm eines 1/2-Faltungscodes mit m=2.

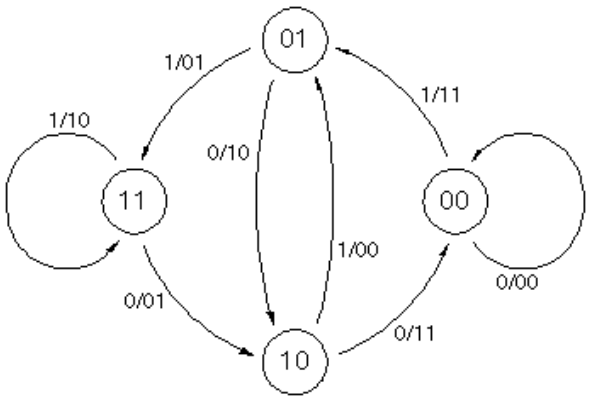

Bild 6.6: Zustandsdiagramm eines 1/2-Faltungscodes mit m=2.

gänge als Kanten, reduziert sich das Netzdiagramm zu einer redundanzfreien Darstellung, dem Zustandsdiagramm (Bild 6.6). An den Kanten werden dann die Eingabe- und Ausgabesymbole angeschrieben. Die Darstellung als gerichteter Graph ist die übersichtlichste Darstellung, wobei die Konstruktion eines Codebaums oder eines Netzdiagramms aus der Graphendarstellung sehr einfach möglich ist.

6.1.1.6 Mathematische Darstellung

Ein Faltungscode lässt sich auch mathematisch beschreiben. Die Eingangsfolge x wird dazu in Vektoren $x = (x_i^{(1)}, x_i^{(2)}, x_i^{(3)}, ..., x_i^{(k)})$, $x_i^{(j)} \in \{0, 1\}$ zu je k Elementen gruppiert. Damit ergibt sich die Eingangsfolge x zu:

$$x = (x_0, x_1, ..., x_i, ...). \tag{6.14}$$

Aus dieser Eingangsfolge wird eine Codefolge y gebildet, die entsprechend in Vektoren

der Länge n gruppiert wird:

$$\begin{aligned} y &= (y_0, y_1, ..., y_i, ...), \\ \text{mit } y_i &= (y_i^{(1)}, y_i^{(2)}, ..., y_i^{(n)}), y_i^{(i)} \in \{0, 1\}. \end{aligned} \tag{6.15}$$

Der Code kann dann durch eine Matrizenmultiplikation beschrieben werden:

$$y = x \cdot G. \tag{6.16}$$

Die Matrix G besteht aus Untermatrizen G_i, $i \in \{0, 1, ..., m\}$, wobei mit m wieder die Gedächtnistiefe bzw. die Anzahl der Vektorregister gemeint ist:

$$G = \begin{pmatrix} G_0 & G_1 & \cdots & G_m & 0 & 0 & 0 & \cdots \\ 0 & G_0 & G_1 & \cdots & G_m & 0 & 0 & \cdots \\ 0 & 0 & G_0 & G_1 & \cdots & G_m & 0 & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \cdots \end{pmatrix} \tag{6.17}$$

Die Untermatrizen G_i sind dabei:

$$G_i = \begin{pmatrix} G_i^{(11)} & \cdots & G_i^{(1n)} \\ \vdots & \ddots & \vdots \\ G_i^{(kl)} & \vdots & G_i^{(kn)} \\ \vdots & \ddots & \ddots \end{pmatrix} \tag{6.18}$$

Die Spaltenvektoren bilden dabei die in Bild 6.2 zu sehenden Koeffizientenvektoren $h_i^{(j)}$, sie definieren sich also mit:

$$h_i^{(j)} = \begin{pmatrix} G_i^{(1j)} \\ \vdots \\ G_i^{(kj)} \end{pmatrix} \tag{6.19}$$

Fasst man für jedes $j \in \{1, 2, ..., n\}$ alle $m+1$ dieser Vektoren wieder zu n neuen Matrizen zusammen, entstehen die so genannten Generatormatrizen H, sie haben folgende Form:

$$H_j = \begin{pmatrix} h_0^{(j)} & h_1^{(j)} & \cdots & h_m^{(j)} \end{pmatrix} = \begin{pmatrix} G_0^{(1j)} & G_1^{(1j)} & \cdots & G_m^{(1j)} \\ G_0^{(2j)} & G_1^{(2j)} & \cdots & G_m^{(2j)} \\ \vdots & \vdots & \ddots & \cdots \\ G_0^{(kj)} & G_1^{(kj)} & \cdots & G_m^{(kj)} \end{pmatrix} \tag{6.20}$$

Die Zeilenvektoren dieser Matrizen sind dann die Impulsantworten des Codierers, sie werden auch als Generatorsequenzen $g_l^{(j)}$ bezeichnet. Sie setzen sich also wie folgt zusammen:

$$\begin{aligned} g_l^{(j)} &= \begin{pmatrix} G_0^{(lj)} & G_1^{(lj)} & \cdots & G_m^{(lj)} \end{pmatrix} \\ &\text{mit } l \in \{1, 2, ..., k\}. \end{aligned} \tag{6.21}$$

Diese Sequenzen werden in der Literatur auch als so genannte Generatorpolynome zur Beschreibung von Faltungscodes aufgelistet. Zu beachten ist dabei, dass diese Sequenzen in oktaler Schreibweise notiert werden und die Elemente der Zeilenvektoren auch in umgekehrter Reihenfolge stehen können (Generatorpolynome für m–Sequenzen).

Beispiel:
Ein Faltungscode mit $R = k/n = 1/4$ und $m = 4$ habe die Generatorpolynome $g^{(0)} = 54_8$, $g^{(1)} = 64_8$, $g^{(2)} = 64_8$ und $g^{(3)} = 74_8$. In Binärdarstellung lautet dann z.B. das Polynom $g^{(0)}$:

$$g^{(0)} = 54_8 = (101100)\,.$$

Daraus lassen sich dann die Koeffizienten $G_i^{(lj)}$ einfach ablesen. Es ergeben sich folgende Werte:

$$G_0^{(10)} = 1, G_1^{(10)} = 0, G_2^{(10)} = 1, G_3^{(10)} = 1, G_4^{(10)} = 0.$$

Da $m = 4$ ist gibt es nur fünf Koeffizienten und nicht sechs, wie aus der binären Darstellung zu vermuten wäre. Es werden an das eigentliche Generatorpolynom einfach so viele Nullen angefügt, bis eine Anzahl von Stellen erreicht ist, die ein ganzzahliges Vielfaches von drei ist. Der Begriff des Generatorpolynoms kann man mit einer weiteren Möglichkeit der mathematischen Formulierung eines Faltungscodes assoziieren. Hankerson synthetisiert Faltungscodes mit Polynomen als Generatoren. Dabei werden die einzelnen Stufen des Schieberegisters mit $X_0, X_1, ..., X_m$ bezeichnet. Die Eingangsdaten liegen an X_0 an. Der Ausgang c zum Zeitpunkt t ist dann:

$$c_t = k_0 X_0(t) + ... + k_m X_m(t) \tag{6.22}$$

mit den Koeffizienten $k_0, ..., k_m$, wobei $k_i \in \{0, 1\}$ gilt, und mit $X_i(t)$ als den Inhalten der Register zum Zeitpunkt t. Das Polynom

$$k(x) = k_0 + k_1 x + k_2 x^2 + ... + k_m x^m. \tag{6.23}$$

ist dann der Generator dieses Schieberegisters. Ihren Ursprung hat die Bezeichnung der Generatoren als Polynome in der den Codes zugrunde liegenden Zahlentheorie. Die mathematischen Operationen der in diesem Abschnitt vorgestellten Formeln sind im Primkörper $GF(2)$ auszuführen (GF: Galois-Feld). Der Primkörper ist wie folgt definiert:
Sei $p \in N$ eine Primzahl, so genügen die Elemente $\{0, 1, ..., p-1\}$ und das Rechnen $(+, \cdot)$ modulo p den Axiomen eines Körpers. Dieser Körper wird dann Primkörper genannt und mit $GF(p)$ bezeichnet.
D.h. für die Addition und die Multiplikation in $GF(2)$ gelten die folgenden Tabellen: Bildet man nun den so genannten Erweiterungskörper zu $GF(p)$ kommt man zu den primitiven Polynomen: Sei $p(x)$ ein irreduzibles Polynom, $gradp(x) = m$, $p_i \in GF(p)$ und sei a eine Wurzel von $p(x)$ (also $p(a) = 0$) mit der Eigenschaft: $a^i \mathrm{mod} p(a), i = 0, 1, ..., p^m - 2$ ergibt genau alle $p^m - 1$ verschiedenen Polynome $f(a) \neq 0$, dann heißt $p(x)$ primitives

+	0	1
0	0	1
1	1	0

Tabelle 6.1: Addition in GF(2).

·	0	1
0	0	0
1	0	1

Tabelle 6.2: Multiplikation in GF(2).

Polynom (a primitives Element). Ein Element des Erweiterungskörpers $GF(p^m)$ wird damit definiert als die Nullstellen a des primitiven Polynoms $p(x)$, nämlich $p(a) = 0$. Die Potenzen des Elementes a, modulo $p(a)$ gerechnet, ergeben den Erweiterungskörper. Daraus resultieren zwei Darstellungsformen für die Elemente des Erweiterunskörpers, die Exponentendarstellung, wo jedes Element als Potenz des primitiven Elements a beschrieben wird, und die Komponentendarstellung, in der jedes Element durch die Koeffizienten der Polynome $f(a)$ beschrieben wird. Schreibt man die Polynome $f(a)$ entsprechend obiger Definition als:

$$f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + ... + f_0,$$

so sind die Polynome durch ihre Koeffizienten bestimmt, und es genügt zu schreiben:

$$\begin{pmatrix} f_{m-1} & f_{m-2} & \cdots & f_0 \end{pmatrix} \qquad f_i \in GF(p)) \,.$$

Dies stellt den Zusammenhang zwischen der Darstellung als Bitvektoren und der Darstellung als Polynome für die Generatoren von Faltungscodes dar. Die Theorie zu den Galois–Feldern und ihre Bedeutung für die Codierung ist in der Literatur weitergehend beschrieben.

6.2 Versuch Faltungscodierer

Zum Einarbeiten soll zunächst ein Faltungscodierer mit Kippgliedern erstellt werden. Der Faltungscodierer soll einen Code mit der Rate $R = k/n = 1/2$ erzeugen und dabei eine Gedächtnistiefe von $m = 2$ haben. Der Code soll mit den Generatoren $g^{(0)} = 5_8$ und $g^{(1)} = 7_8$ erzeugt werden (die Koeffizienten des Eingangsbits sind die LSB der Generatoren). Gehen Sie dabei wie folgt vor:

1. Angabe des Projektverzeichnisses (zur Vermeidung späterer Probleme sollte für jedes Projekt ein eigenes Verzeichnis angelegt werden), Angabe des Projektnamens und des Namens der obersten Hierarchiestufe des Entwurfs (diese sind normal identisch mit dem Verzeichnisnamen).

2. Angabe von Dateien, die dem Projekt hinzugefügt werden (für einfache Projekte keine).

3. Spezifizieren von externen Entwurfsprogrammen (trifft für das Praktikum nicht zu).

4. Angabe der Bausteinfamilie (hier: APEX20KE und Zuweisung eines spezifischen Bausteins auswählen).

5. Auswahl des Bausteins: Typ EP20K30ETC144-1X (mit 144 Pins, TQFP–Package und speed–grade: Fastest) verwendet. Der Grund hierfür liegt im verfügbaren und übersichtlichen Timing Closure Floorplan des Bausteins!

6. Dokumentieren Sie Ihre Schaltungen und Ergebnisse für den Praktikumsbericht!

6.2.1 Encoder mit Schieberegister erstellen

Dieser Encoder kann mit einem m–stufigen Schieberegister und Antivalenz–Verknüpfungen realisiert werden.

- Erstellen Sie den oben angegebenen Faltungscodierer grafisch in einem Projekt. Benutzen Sie dazu Kippglieder und Verknüpfungen Ihrer Wahl. Die Schaltung soll einen Dateneingang, einen Takteingang, einen Rücksetzeingang für die Kippglieder und zwei Ausgänge haben. Der Rücksetzeingang soll H–aktiv sein, die Eingansdaten sollen von Ihrer Schaltung mit der steigenden Taktflanke übernommen werden.

- Compilieren Sie die Schaltung.

- Entwerfen Sie einen Simulationsstimuli (achten Sie auf korrektes Rücksetzen der Schaltung am Anfang der Simulation). Ihr Stimuli soll mindestens die Datenfolge $B9_h$ enthalten (MSB zuerst).

- Simulieren Sie die Schaltung.

- Diskutieren Sie ob die Schaltung mit sechs Elementen (Ein- u. Ausgänge ausgenommen) realisierbar ist.

6.2.2 Encoder mit VHDL erstellen

Der Codierer lässt sich auch als Zustandsdiagramm darstellen.

- Starten Sie ein neues Projekt.

- Entwerfen Sie einen Zustandsautomaten in VHDL der diesen Code realisiert. Benutzen Sie dazu ein PROCESS–Konstrukt. Die Rücksetzbedingung soll mit einer IF–THEN... Bedingung erkannt werden, der Zustandswechsel mit CASE–WHEN Strukturen.

- Compilieren Sie den Entwurf.
- Betrachten Sie das Ergebnis der Timing–Analyse und die Realisierung im Floorplan, achten Sie dabei auf die Laufzeiten der einzelnen Verbindungen.
- Platzieren Sie die gesamte Schaltung im Floorplan in LAB_A1.
- Compilieren Sie den Entwurf nochmals.
- Diskutieren Sie die Realisierung im Floorplan und die Laufzeiten.

Falls Sie gute VHDL Kenntnisse besitzen, versuchen Sie diese Aufgabe mit parametrisierten Werten für die Generatoren und die Gedächtnistiefe zu erstellen.

6.2.3 Encoder mit AHDL erstellen

- Erstellen Sie ein neues Projekt.
- Realisieren Sie den Codierer in AHDL. Benutzen Sie dabei eine ähnliche Struktur wie in VHDL (d.h. CASE–WHEN Struktur).
- Starten Sie die Compilierung.
- Lassen Sie ein Block Symbol File automatisch erstellen.

6.2.4 Vergleich der Entwürfe

- Erstellen Sie für die beiden ersten Entwürfe jeweils ein Block Symbol File.
- Erstellen Sie ein neues Projekt.
- Kopieren Sie die Entwurfsdateien der vorigen Projekte (*.bdf, *.vhd, *.tdf) und die erstellten Symbol Dateien (*.bsf) in einen neuen Unterordner des neuen Projekts.
- Fügen Sie die Entwurfsdateien zum aktuellen Projekt hinzu.
- Erstellen Sie ein neues Schematic Design File und fügen Sie die Blocksymbole der vorigen Projekte ein.
- Verbinden Sie alle Dateneingänge mit einem gemeinsamen Eingang, verfahren Sie so für den Takt und das Rücksetzsignal. Erstellen sie separate Ausgänge, wobei die Ausgänge der einzelnen Blöcke als Bus ausgeführt werden soll.
- Platzieren Sie die einzelnen Blöcke jeweils im ersten LAB der Zeilen A, B und C.
- Compilieren Sie die gesamte Schaltung.
- Erstellen Sie einen Stimulus und simulieren Sie die Schaltung.
- Diskutieren Sie das Ergebnis. Achten Sie auf Unterschiede der Signale.

7 Projekt Lauflicht

Zum Erproben des Bausteins auf dem Altera Entwicklungs–Board soll eine kleine Schaltung erstellt werden, die die Leuchtdioden auf der Platine ansteuert. Verwenden Sie für die Erstellung Ihres Projektes das Device EP2C70F672C6. Setzen Sie alle nicht verwendeten Pins auf tri–state!

- Erstellen Sie eine Schaltung, die einen Lichtpunkt erzeugt, der von links nach rechts und zurück läuft. Steuern Sie die LEDs so an, dass der Wechsel des Lichtpunkts gut zu erkennen ist (der Takt des Bausteins beträgt 100 Mhz). Die Realisierung der Schaltung bleibt Ihnen überlassen (z.B. Ringschieberegister und Multiplexer oder Zustandsautomat mit VHDL ...).
- Verwenden Sie die auf dem Entwicklungs–Board vorhandene Sieben–Segment Anzeige zum zählen der Durchläufe.
- Programmieren Sie eine variable Laufgeschwindigkeit des Lauflichtes. Verwenden Sie dazu die Taster oder den DIP–Schalter des Entwicklungs–Boards.
- Verwenden Sie den Dippschalter als Eingabemöglichkeit eines Musters, welches dann als Lauflicht durchgeschoben wird.
- Definieren Sie Start, Abbruch, Reset und Musterübernahme per Taster.
- Arbeiten Sie möglichst mit einem modularen Projekt.
- Weisen Sie Ihre Ausgänge entsrechend dem Datenblatt des Entwicklungs–Boards den Anschlüssen des Bausteins zu.
- Compilieren und simulieren Sie Ihren Entwurf.
- Programmieren und testen Sie Ihre Schaltung.
- Dokumentieren Sie Ihre Schaltung für den Praktikumsbericht (Screenshots, Ergebnisse!).

8 Projekt Digitale Filter

8.1 Theorie

8.1.1 Projektverwaltung

8.1.1.1 Verwendung von Generics

Generics dienen dazu, statische Informationen von der Umgebung an einen Block weiterzugeben. Es gibt keinerlei Typbeschränkungen. Allerdings darf ein deklariertes Element nicht zur Berechnung eines anderen Elements herangezogen werden. Der nachfolgende Quelltextauszug zeigt einige Beispiele. Besitzt eine Generic–Deklaration keinen Standardwert, so muss bei der Instantiierung diesem Generic unbedingt ein Wert zugewiesen werden. Somit empfiehlt es sich immer einen Standardwert vorzugeben.

```
Generic(fir_ordnung : Natural := 63; -- Generic mit Standardwert
        bit_da : Natural; -- Generic ohne Standardwert
        bit_ad : Natural := bit_da; -- illegal!
        bsp_str : String := "‘Beispiel"’) -- Text als Generic - zulässig
```

8.1.1.2 Verwendung von eigenen Packages

Die Einbindung von Standardpackages gestaltet sich problemlos, selbst bei einem hierarchischen Entwurf. Dies ist bei Verwendung eigener Packages leider nicht der Fall. Diese müssen immer als Datei in das jeweilige Quartus–Projekt und auch in das übergeordnete Projekt eingebunden werden. Es genügt somit nicht, sie ins Sub–Verzeichnis zu kopieren, welches als User–Library deklariert ist. Dies ist besonders verwirrend, wenn in einem übergeordneten Projekt das Package selbst nicht verwendet wird, sondern nur in einem untergeordneten Teilentwurf. Es empfiehlt sich, die Dateien mit eigenen Packages an oberster Stelle im Projekt einzufügen, da die Projektdateien der Reihe nach abgearbeitet werden.

8.1.2 Grundlagen der digitalen Signalverarbeitung

In diesem Kapitel werden einige Grundlagen der digitalen Signalverarbeitung dargelegt. Zuerst werden die Prinzipien und Probleme der Wandlung vom analogen Signal zur digitalen Repräsentation und umgekehrt aufgezeigt. Eine Betrachtung diskreter Signale und Systeme, mit besonderem Fokus auf den Beschreibungsmöglichkeiten, schließt sich an. Danach folgt eine eingehende Erörterung möglicher Zahlendarstellungen in einem digitalen System. Der Schwerpunkt liegt dabei auf den Festkommazahlen und Rechenoperationen in diesem Format.

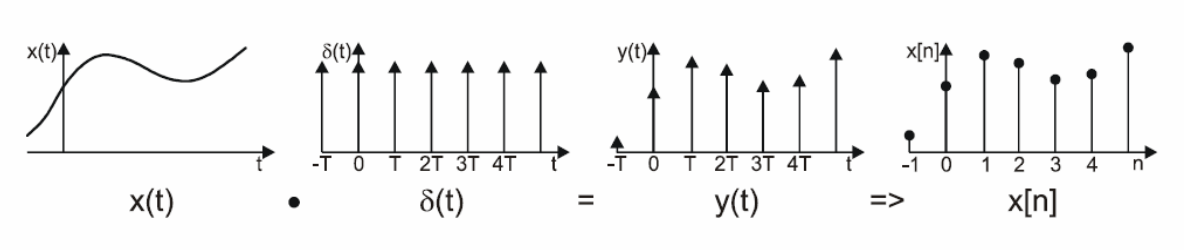

Bild 8.1: Von der kontinuierlichen Funktion zu den Abtastwerten.

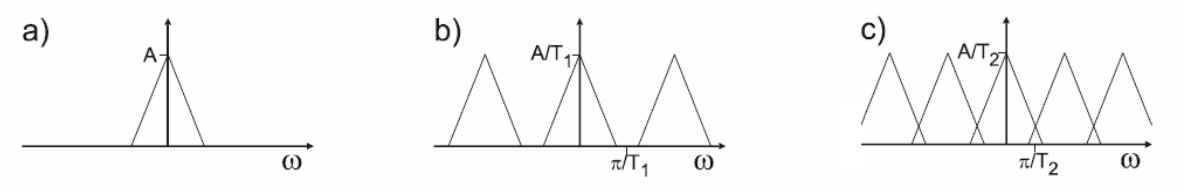

Bild 8.2: a) Spektrum eines kontinuierlichen Signals, b) Spektrum des abgetasteten Signals mit $1/2\ f_a > f_{max}$ – keine Überlappung und c) Spektrum des abgetasteten Signals mit $1/2\ f_a < f_{max}$ – Überlappung (Aliasing).

8.1.2.1 A/D– und D/A–Wandlung

In diesem Abschnitt werden die Grundlagen der Signalwandlung vom Analogen ins Digitale und umgekehrt beschreiben. Ausführliche Erklärungen finden sich in vielen Büchern über die digitale Signalverarbeitung.

8.1.2.1.1 Ideale Abtastung des analogen Signals

Unter idealer Abtastung versteht man die Multiplikation des analogen Eingangssignals $x(t)$ mit einer Folge von Diracstössen mit konstantem Zeitabstand T, dem Abtastintervall. Es entsteht die zeitkontinuierliche Funktion $y(t)$ (Bild 8.1). Unter Ausnutzung der Ausblendeigenschaft des Diracstosses lässt sich diese als Folge diskreter Abtastwerte, die dem Gewicht des jeweiligen Diracstosses entsprechen, darstellen. Ein solches Signal wird als diskretes Signal bezeichnet. Diese Werte sind zwar zeitdiskret aber noch amplitudenkontinuierlich. Damit sie sich in einem digitalen System verarbeiten lassen, werden sie quantisiert.

8.1.2.1.2 Das Abtasttheorem

Berechnet man das Spektrum des abgetasteten Signals, so erkennt man, dass dieses die periodische Fortsetzung des ursprünglichen analogen Spektrums, gewichtet mit dem Faktor $1/T$, ist. Die Periodendauer beträgt $2\pi/T$. Tritt zwischen der Grundperiode und den folgenden Perioden keine Überlappung auf (Bild 8.2), so kann das kontinuierliche Signal exakt rekonstruiert werden. Treten dagegen Überschneidungen auf, wie in Bild 8.2 zu sehen, so kann das ursprüngliche Spektrum nicht mehr durch einen Tiefpass rekonstruiert werden. Man bezeichnet dies als Aliasing. Nach dem Abtasttheorem von Shannon kann Überlappungsfreiheit nur dann bestehen, wenn die halbe Abtastfrequenz

größer als die Bandbreite des Signals ist. Bei Tiefpasssignalen ist die Bandbreite gleich der höchsten vorkommenden Frequenz.

8.1.2.1.3 Signalrekonstruktion

Zur Rekonstruktion des analogen Signals muss man das abgetastete Signal mit einem idealen Tiefpass der Grenzfrequenz $1/2f_a$ filtern, wie das Spektrum in Bild 8.2 zeigt. Die Übertragungsfunktion $H(\omega)$ und die daraus resultierende Impulsantwort $h(t)$ dieses Filters lauten:

$$H(\omega) = \begin{cases} T & \text{für } |\omega| < \frac{\pi}{T} \\ 0 & \text{für } |\omega| > \frac{\pi}{T} \end{cases} \quad \rightleftharpoons \quad h(t) = \frac{\sin\frac{\pi t}{T}}{\frac{\pi t}{T}} \tag{8.1}$$

Das ursprüngliche Signal wird im Zeitbereich aus der Faltung der gewichteten Diracstösse (Abtastwerte) mit der Impulsantwort des Rekonstruktionsfilters bestimmt. Dies entspricht der Summe von unendlich vielen gegeneinander verschobenen sin(x)/x–Funktionen, die mit den Abtastwerten gewichtet sind. Dies ist in Bild 8.3 illustriert. Der ideale Rekonstruktionsfilter wird als Cardinal Hold Filter bezeichnet.

8.1.2.1.4 Praktische Realisierung

Die praktische Realisierung der Zeitdiskretisierung ist mittels Sample&Hold–Schaltung ohne Verletzung der theoretischen Annahmen (Diracstoss) möglich. Es ergeben sich lediglich Fehler aus den nicht genau im gleichen Abstand liegenden Abtastzeitpunkten, dem so genannten Jitter, und der Quantisierung der Amplitude auf eine bestimmte Bitbreite. Die Rekonstruktion bereitet größere Probleme. Wie im Kap. 8.1.2.1.3 gezeigt, verlangt die ideale Rekonstruktion die Addition von unendlich vielen akausalen Funktionen, was in der Praxis natürlich unmöglich ist. Die meisten D/A–Wandler arbeiten mit einem so genannten Zero–Order Hold Filter. Dieser besagt nichts anderes, als dass der aktuelle Abtastwert als Funktionswert bis zum nächsten Wert beibehalten wird. Es entsteht also eine Treppenfunktion, die ihren Funktionswert gerade an den Abtastzeitpunkten ändert. Die Impulsantwort und Übertragungsfunktion ergeben sich zu:

$$h(t) = \begin{cases} 1 & \text{für } 0 < t < T \\ 0 & \text{sonst} \end{cases} \quad \rightleftharpoons \quad H(f) = T \cdot \frac{\sin \pi f T}{\pi f T} \cdot e^{j\pi f T} \tag{8.2}$$

Man erkennt eine $\sin(\pi fT)/\pi fT$–Verzerrung im Frequenzband. Bei Anwendungen in denen die Verzerrung nicht toleriert werden kann, muss diese entweder im folgenden analogen Filter oder schon vorher im digitalen Filter kompensiert werden. Zeichnet man die Funktionsspektren vor und nach der Filterung qualitativ auf (Bild 8.4), so zeigt sich, dass zusätzlich noch Frequenzanteile oberhalb der halben Abtastfrequenz vorkommen. Diese Frequenzen müssen vom nachfolgenden Analogfilter ausgeblendet werden. Weitere Rekonstruktionsvarianten sind in der einschlägigen Literatur zu finden, sie sind für dieses Praktikum aber ohne Bedeutung, da der D/A–Wandler im Praktikum mit dem Zero-Order Hold Verfahren arbeitet.

8.1.2.2 Diskrete Signale

Ein diskretes Signal ist nur zu diskreten Zeitwerten, die im gleichen Abstand liegen,

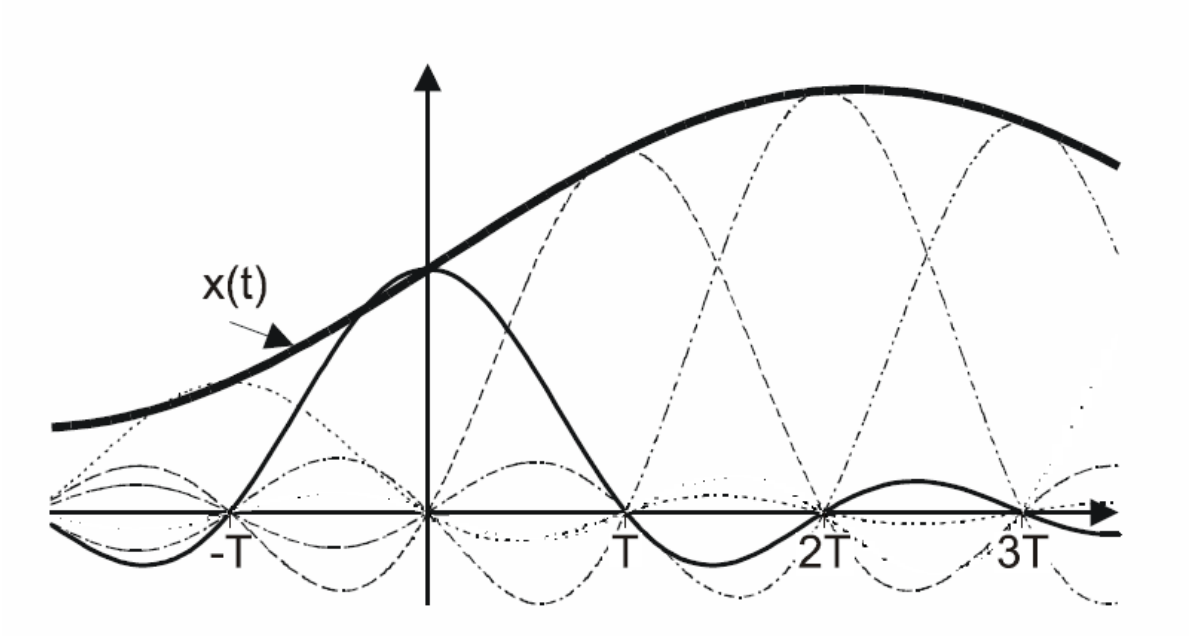

Bild 8.3: Signalrückgewinnung mittels idealem Tiefpass.

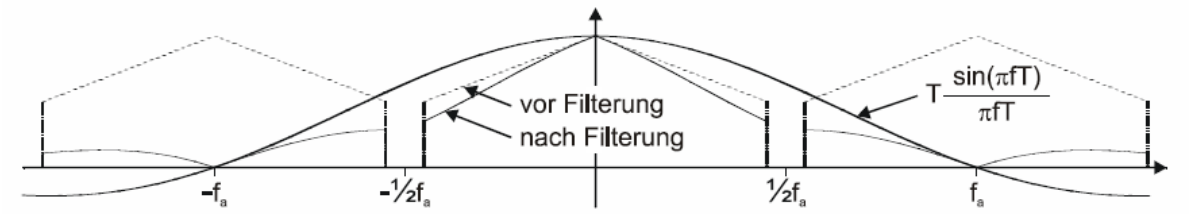

Bild 8.4: Qualitative Spektren vor und nach Zero–Order Hold Filterung.

definiert. Das Ergebnis einer A/D–Wandlung ist z.B. ein solches Signal. Signale lassen sich allgemein entweder im Zeitbereich oder im Frequenzbereich beschreiben. Beide Arten sind relativ einfach ineinander zu Überführen und mal ist die eine Sichtweise praktikabler, mal die andere.

8.1.2.2.1 Beschreibung im Zeitbereich

Im Zeitbereich wird ein diskretes Signal durch seinen Funktionswert zu jedem diskreten Zeitwert nT beschrieben. Dabei ist T der Abstand zwischen zwei aufeinander folgenden Werten, auch Abtastintervall genannt, und $f = 1/T$ die Abtastfrequenz. In Formel 8.3 ist als Beispiel eine diskrete Sinusfunktion aufgeführt. Die Schreibweise $x[n]$ ist eine gängige Abkürzung von $x[nT]$, und wird im weiteren Verlauf verwendet.

$$x[n] = 2 \cdot \sin(n \cdot \omega t) \tag{8.3}$$

Es gibt einige wichtige diskrete Signale, die sehr häufig benutzt werden. Dazu zählt der diskrete Einheitsimpuls, wie er in Formel 8.4 aufgeführt ist. Er entspricht dem Diracim-

puls der kontinuierlichen Signale, ohne dessen Problem der unendlichen Energiedichte zu teilen.

$$\delta[n] = \begin{cases} 1 & \text{für } n = 0 \\ 0 & \text{für } n \neq 0 \end{cases} \tag{8.4}$$

Sehr häufig wird dieser Impuls in seiner um i Zeitintervalle verschobenen Form, wie er in Formel 8.5 angegeben ist, gebraucht.

$$\delta[n-i] = \begin{cases} 1 & \text{für } n = i \\ 0 & \text{für } n \neq i \end{cases} \tag{8.5}$$

Mit Hilfe dieses verschobenen Einheitsimpulses können diskrete Signale in anderer Weise beschrieben werden. Formel 8.6 zeigt diese oft verwendete Schreibweise.

$$x[n] = \sum_{i=-\infty}^{\infty} x[i] \cdot \delta[n-i] \tag{8.6}$$

Zu guter Letzt sei noch der diskrete Einheitssprung in Formel 8.7 erwähnt. Dieses Signal wird uns wie der Einheitsimpuls bei der Filtersimulation wieder begegnen.

$$u[n] = \begin{cases} 1 & \text{für } n < 0 \\ 0 & \text{für } n \geq 0 \end{cases} \tag{8.7}$$

8.1.2.2.2 Beschreibung im Frequenzbereich

Kontinuierliche Signale lassen sich mittels der Fouriertransformation im Frequenzbereich beschreiben. Diese Transformation lässt sich auch auf diskrete Signale anwenden. Berücksichtigt man die speziellen Eigenschaften der diskreten Signale, so ergibt sich die Fouriertransformation diskreter Signale (DFT) nach Formel 8.8.

$$X[\omega] = \sum_{i=-\infty}^{\infty} x[n] \cdot e^{-j\omega nT} \tag{8.8}$$

Das Ergebnis ist eine periodische Funktion mit der Periode $2\pi/T$, was schon im Kap. 8.1.2.2.2 zur Herleitung des Abtasttheorems erwähnt wurde. Es genügt somit, das Basisintervall von $\omega = -\pi/T...\pi/T$ zu betrachten. Häufig wird mit der normierten Kreisfrequenz $\Theta = \omega T$ gearbeitet.

8.1.2.2.3 Digitale Signale

Digitale Signale sind ein Spezialfall der diskreten Signale. Während bei Letzteren die Funktionswerte kontinuierlich jeden beliebigen Wert annehmen können, sind digitale Signale quantisiert. Ihre Werte werden mit einer endlichen Menge von Bit codiert und sind somit ebenfalls diskretisiert.

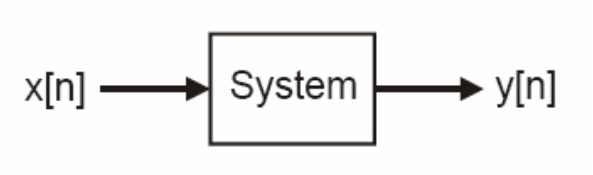

Bild 8.5: Blockschaltbild eines diskreten Systems.

8.1.2.3 Diskrete Systeme

Unter einem diskreten System versteht man eine Einheit, die aus einer bestimmten Anzahl von diskreten Eingangssignalen unter Verwendung definierter Regeln eine bestimmte Anzahl von diskreten Ausgangssignalen erzeugt. Wir beschränken uns auf die Betrachtung der Klasse der linearen zeitinvarianten diskreten Systeme (LTD–Systeme) mit je einem Eingangs- und Ausgangssignal. Bild 8.5 zeigt das Blockschaltbild eines solchen Systems. Eine wichtige Eigenschaft realisierbarer Systeme ist deren Kausalität. Ein System ist kausal, wenn sein Ausgang nur vom aktuellen und von vorhergehenden Eingangswerten und von vergangenen Ausgangswerten, nicht aber von zukünftigen Eingangswerten abhängt. Die Stabilität von Systemen spielt ebenfalls eine wichtige Rolle. Ein stabiles System bedeutet, dass ein amplitudenbegrenztes Eingangssignal ein amplitudenbegrenztes Ausgangssignal zur Folge hat.

8.1.2.3.1 Beschreibung im Zeitbereich

Eine der Möglichkeiten, ein System im Zeitbereich zu beschreiben, ist die Darstellung mit Differenzengleichungen. Diese beschreiben den Wert eines diskreten Signals zu einem bestimmten Zeitpunkt als Funktion von Signalwerten des gleichen Zeitpunkts oder vorangegangener Zeitpunkte. In Formel 8.9 ist als Beispiel eine Differenzengleichung erster Ordnung gegeben.

$$y[n] = a \cdot x[n] + b \cdot x[n-1] + c \cdot y[n-1] \tag{8.9}$$

Beschränkt man sich auf ein LTD–System, so kann dieses allgemein durch folgende Gleichung beschrieben werden:

$$y[n] = \sum_{i=0}^{N} b_i \cdot x[n-1] + \sum_{l=1}^{M} a_l \cdot y[n-1] \tag{8.10}$$

Eine weitere Möglichkeit, ein System komplett zu beschreiben, besteht darin seine Impulsantwort anzugeben. Diese bekommt man bei diskreten Systemen durch Anlegen des diskreten Einheitsimpulses (Formel 8.4) und Aufzeichnen des Ausgangsignals. Die Systemantwort auf ein beliebiges Signal lässt sich als Faltung der Impulsantwort mit diesem Signal bestimmen und ergibt sich somit zu:

$$y[n] = h[n] * x[n] = \sum_{i=-\infty}^{\infty} h[i] \cdot x[n-1] \tag{8.11}$$

In vielen Fällen lässt sich die Impulsantwort sehr einfach bestimmen, z.B. ist sie bei FIR–Filtern gerade die Folge der Koeffizienten. Aus der Impulsantwort lässt sich einfach auf die Stabilität und Kausalität des Systems schließen.

Stabilitätsbedingung:

$$\sum_{i=-\infty}^{\infty} |h[i]| < \infty \tag{8.12}$$

Kausalitätsbedingung:

$$h[n] = 0 \quad \text{für} \quad n < 0 \tag{8.13}$$

8.1.2.3.2 Beschreibung im Frequenzbereich

Überführt man Formel 8.11 mittels der diskreten Fouriertransformation in den Frequenzbereich, so erhält man:

$$Y(\omega) = H(\omega) \times X(\omega) \tag{8.14}$$

Dabei wird $H(\omega)$ als Übertragungsfunktion bezeichnet. Die Faltung im Zeitbereich hat sich zu einer Multiplikation im Frequenzbereich gewandelt. Da $H(\omega)$ die Fouriertransformierte der Impulsantwort ist, wird das System komplett durch die Übertragungsfunktion beschrieben. Besonders bei digitalen Filtern hat sie große Aussagekraft, da sich der Amplituden- und Phasengang direkt aus ihr ableiten lässt.

8.1.2.3.3 Die Z–Transformation

Für diskrete Systeme von entscheidender Bedeutung ist die Z–Transformation. Sie überführt Signale und Systeme vom diskreten Zeitbereich in den komplexen Frequenzbereich. Dies wird durch Einführung der komplexen Frequenzvariablen z erreicht. Die Transformationsvorschrift für ein diskretes Signal ist in Formel 8.15 wiedergegeben.

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] \cdot z^{-n} \tag{8.15}$$

Ein Vorteil der Z–Transformation liegt darin, dass man aus dem Blockschema eines Systems meist direkt die Z–Transformierte der Impulsantwort, welche man Systemfunktion nennt, ablesen kann. Betrachtet man die Definition der DFT (Formel 8.8), so erkennt man, dass die Übertragungsfunktion durch Variablensubstitution von $z^{-1} = e^{-j\omega T}$ aus der Systemfunktion bestimmt werden kann. Dies ist nichts anderes als der Einheitskreis der komplexen Z–Ebene. Die allgemeine Beschreibung eines LTD–Systems haben wir in Formel 8.10 kennen gelernt. Daraus ergibt sich die Systemfunktion zu:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{N} b_i \cdot z^{-i}}{1 - \sum_{l=1}^{M} a_l \cdot z^{-l}} \tag{8.16}$$

Eine eingehende Beschreibung der Z–Transformation findet sich in der Literatur.

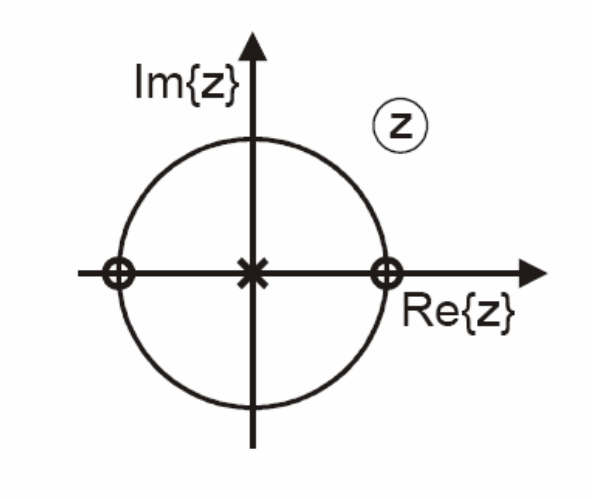

Bild 8.6: Pol–/Nullstellendarstellung.

8.1.2.3.4 Pole und Nullstellen als Systembeschreibung

Formel 8.16 lässt sich in folgende Form bringen:

$$H(z) = c_0 \cdot \frac{(z-n_1)\cdot(z-n_2)...(z-n_N)}{(z-p_1)\cdot(z-p_2)...(z-p_M)} \cdot z^{M-N} \tag{8.17}$$

Dabei werden die komplexen Zahlen n_1 bis n_N als Nullstellen, p_1 bis p_M als Pole von $H(z)$ bezeichnet. Die Systemfunktion lässt sich somit bis auf die Konstante c_0 komplett durch ihre Pole und Nullstellen darstellen. Dies ist also eine weitere Möglichkeit, ein System zu beschreiben. In der komplexen Z–Ebene lassen sich die Pol- und Nullstellen grafisch darstellen. Üblicherweise wird ein Pol durch ein Kreuz und eine Nullstelle durch einen Kreis markiert. Als Beispiel ist in Bild 8.6 das Pol-/Nullstellendiagramm eines FIR–Bandpasses zweiter Ordnung abgebildet. Aus einer solchen Darstellung lassen sich viele Systemeigenschaften ablesen. Hier sei als einzige Eigenschaft erwähnt, dass stabile Systeme nur Pole innerhalb des Einheitskreises besitzen. Für weitergehende Untersuchungen sei auf die Literatur verwiesen.

8.1.2.4 Zahlendarstellung

In digitalen Systemen können nur Zahlenwerte verarbeitet werden, die durch eine bestimmte Anzahl von Bits, einem so genannten Wort, dargestellt werden. Dieses Kapitel beschreibt verschiedene gängige Interpretationen dieser Bitfolgen. Insbesondere wird auf die Festkommadarstellung und deren Rechenoperationen eingegangen, da alle in dieser Arbeit entwickelten Filter mit dieser Darstellung arbeiten. Kenntnisse der Zahlenbasiswandlung (binär $\leftrightarrow$ dezimal) werden vorausgesetzt. Diese Thematik wird in der Literatur ausführlich beleuchtet. Bild 8.7 zeigt eine gängige Indizierung der einzelnen Bits eines Wortes der Länge n, wie sie im weiteren Verlauf verwendet wird. Bewusst ist dieses Kapitel nahe an den Versuchsanforderungen des Praktikums gehalten und theoretische Abhandlungen weitestgehend vermieden worden.

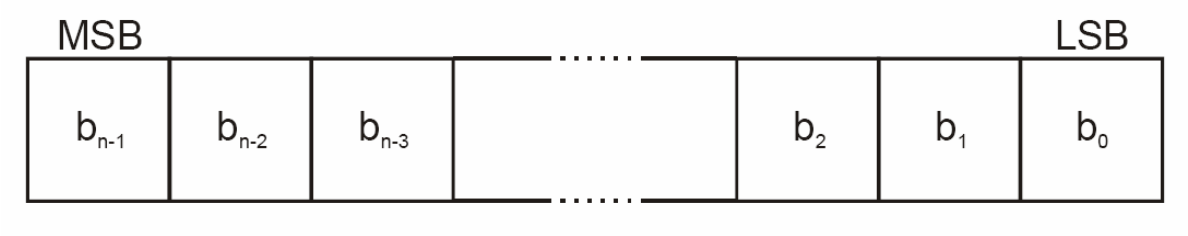

Bild 8.7: Bezeichnung der Bits eines Wortes.

8.1.2.4.1 Natürliche Zahlen

Eine natürliche Zahl wird als Bitkombination dargestellt, indem man per Definition dem niederwertigsten Bit (LSB) die Wertigkeit 2^0, dem zweiten Bit 2^1, usw. zuweist. Das höchstwertige Bit (MSB) hat bei einer Zahl aus n Bits somit die Wertigkeit 2^{n-1}. In Formel 8.18 ist angegeben, wie sich der Zahlenwert allgemein ergibt.

$$x = \sum_{i=0}^{n-1} b_i \cdot 2^i \tag{8.18}$$

Eine solche Zahl kann durch das hinzufügen von Null–Bits vor dem MSB auf eine größere Bitbreite gebracht werden.

8.1.2.4.2 Positive Festkommazahlen

Fügt man analog zum Dezimalsystem ein Komma an eine bestimmte Position der Binärzahl, so ergibt sich eine gebrochen rationale Zahl. Hat diese Zahl I Stellen vor dem Komma und Q Stellen danach, so spricht man von einer I, Q–Darstellung. In Formel 8.19 ist wiederum der Zahlenwert angegeben.

$$x = \sum_{i=0}^{I-1} b_{n-Q+i} \cdot 2^i + \sum_{i=1}^{Q} b_{Q-i} \cdot 2^{-i} = \sum_{i=0}^{n} b_i \cdot 2^{i-Q} \tag{8.19}$$

Solche Zahlen können entweder durch hinzufügen von M Bit vor dem MSB, dies ergibt dann eine $I + M, Q$–Darstellung, oder durch anhängen von L Bit nach dem LSB ($I, Q + L$–Darstellung) erweitert werden. Allerdings lassen sich nur 2^{I+Q} Zahlen genau darstellen. Andere rationale Zahlen können nur als Näherung codiert werden. Der Abstand zwischen zwei benachbarten Zahlen beträgt immer 2^{-Q} über den gesamten darstellbaren Zahlenraum, somit beträgt der maximale absolute Fehler bei Rundung $2^{-(Q+1)}$. Der maximale *relative* Fehler der codierten Zahl zur ursprünglichen variiert stark innerhalb des Wertebereichs und ist an der oberen Grenze am geringsten.

8.1.2.4.3 Zweierkomplementdarstellung

Bis jetzt haben wir nur positive Zahlen betrachtet. Will man auch negative darstellen, so könnte man vereinbaren, dass das erste Bit einer Zahl das Vorzeichen darstellt („0“ für positiv – „1“ für negativ) und den Rest wie gewohnt interpretieren. Diese Vorgehensweise hat allerdings einige Nachteile wie z.B. die Doppeldeutigkeit der Zahl Null.

Binärdarstellung von 11	0	1	0	1	1
Bits invertieren (Einserkopmplement)	1	0	1	0	0
LSB addieren	0	0	0	0	1
Ergebnis: Zweierkomplement	1	0	1	0	1

Tabelle 8.1: Zweierkomplementbildung am Beispiel -11 bei 5 Bit.

	6	-6
5 Bit	00110	11010
8 Bit	00000110	11111010

Tabelle 8.2: Bitbreitenänderung einer Zweierkomplementzahl.

Deshalb hat sich die Zweierkomplementdarstellung in den meisten digitalen Systemen durchgesetzt. Dabei erhält man die negativen Zahlen durch bitweises invertieren des Betrages und anschließender Addition eines LSB, wie in Tabelle 8.1 am Beispiel der Zahl -11 bei 5 Bit Ganzzahlendarstellung demonstriert wird. Diese Vorgehensweise wird als Zweierkomplementbildung bezeichnet. Die Vorteile dieser Darstellung sind eine einfache Subtraktion, sie kann durch eine Addition ersetzt werden indem der Subtrahend durch sein Zweierkomplement ersetzt wird. Des weiteren existieren für die Multiplikation effiziente Algorithmen. Eine der wichtigsten Eigenschaften bei Zweierkomplementdarstellung ist das Überlaufverhalten. Addiert man zur größten darstellbaren positiven Zahl ein LSB so erhält man die größte darstellbare negative Zahl. Umgekehrt erhält man bei einem negativen Überlauf die größte positive Zahl. Dieser Überlauf wird im nächsten Kapitel genauer behandelt. Beim erweitern einer solchen Zahl auf eine größere Bitbreite ist zu beachten, dass die zusätzlichen Bits vor dem MSB jetzt nicht mit Nullen, sondern mit dem Inhalt des MSB (dem Vorzeichenbit) gefüllt werden müssen, wie dies in Tabelle 8.2 für die beiden Zahlen 6 und -6 illustriert ist.

8.1.2.4.4 Festkommadarstellung

Unter Festkommadarstellung versteht man die Darstellung im I, Q–Format, wie bereits in Kap. 8.1.2.4.2 beschrieben, allerdings in Zweierkomplementdarstellung. Wir beschränken uns hier meist der Einfachheit wegen auf die Betrachtung des $1, Q$–Formates, da jedes andere Format derselben Bitlänge ($I, Q - I + 1$–Format) daraus durch Multiplikation mit 2^x ($x = I - 1$) erzeugt werden kann. In den Versuchen wird ebenfalls nur die Festkommadarstellung im $1, Q$–Format vorkommen. Der Unterschied zwischen zwei benachbarten darstellbaren Zahlen beträgt $\Delta = 2^{-Q}$ und der Zahlenbereich erstreckt sich von -1 bis $1 - 2^{-Q}$. Die Zahl 1 selbst lässt sich *nicht* darstellen. Bild 8.8 zeigt die Quantisierungskennlinie einer $1, 2$ Bit Festkommazahl. Das Überlaufverhalten, welches schon im vorherigen Kapitel erwähnt wurde, wird auch Modulo 2 Verhalten genannt, da sich die Werte im Abstand von 2 wiederholen. Eine wichtige Eigenschaft dieses Modulo 2 Verhaltens ist, dass Teilergebnisüberläufe im Endergebnis keine Rolle spielen. In Bild 8.9

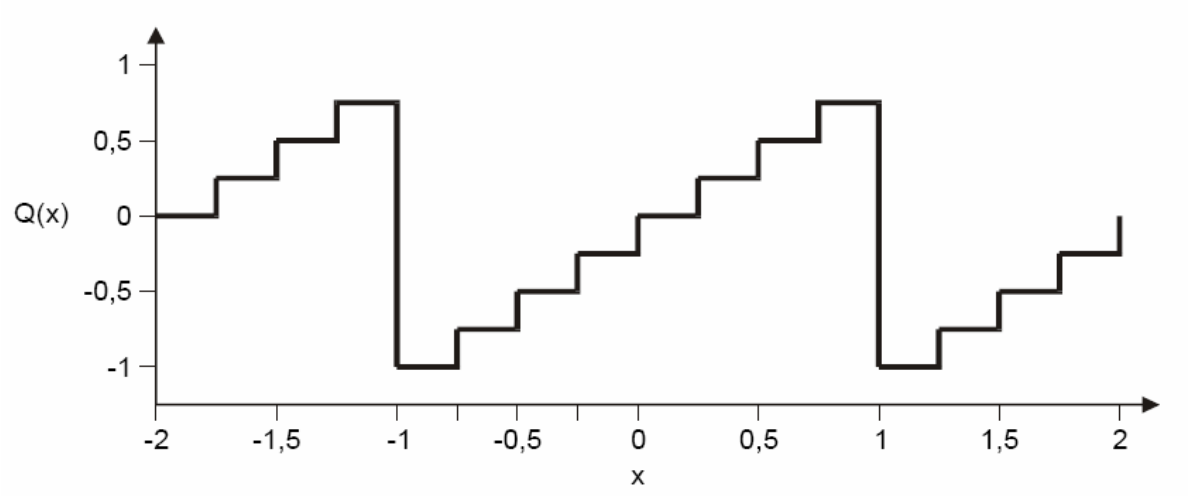

Bild 8.8: Quantisierungskennlinie mit Modulo 2 Überlauf.

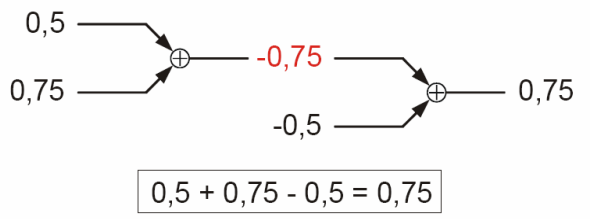

Bild 8.9: Teilergebnisüberlauf einer 1,2 Zahl.

ist dies an einem Rechenbeispiel demonstriert. Die Teilsumme von $0,5+0,75$ ergibt 1,25, was einen Überlauf erzeugt und nach der Quantisierungskennlinie von Bild 8.8 zum Wert $-0,75$ führt. Es entsteht somit ein falsches Zwischenergebnis. Nach der Subtraktion von 0,5 liegt das Endergebnis allerdings wieder im gültigen Bereich und beträgt korrekterweise $0,75$. Allerdings muss man sicher sein, dass das Endergebnis im gültigen Bereich liegt. Bei den in den Praktikumsversuchen behandelten FIR–Filtern kann dies z.B. durch Normierung der Konstantenbetragssumme auf eins garantiert werden. Bei rückgekoppelten Systemen wie IIR–Filter können solche Teilüberläufe allerdings verheerende Folgen haben. Dort wird dann meist mit der Sättigungskennlinie nach Bild 8.10 gearbeitet.

8.1.2.5 Rechenoperationen in Festkommadarstellung

Dieses Kapitel behandelt die drei Grundrechenarten Addition, Subtraktion und Multiplikation und die Bitbreitenänderung ausführlich, um einen leichten Einstieg zu gewährleisten.

8.1.2.5.1 Bitbreitenänderung

Die Bitbreitenänderung ist unter den Rechenoperationen aufgeführt, da sie meist im Zusammenhang mit einer solchen benötigt wird. Zum Beispiel werden beide Summanden

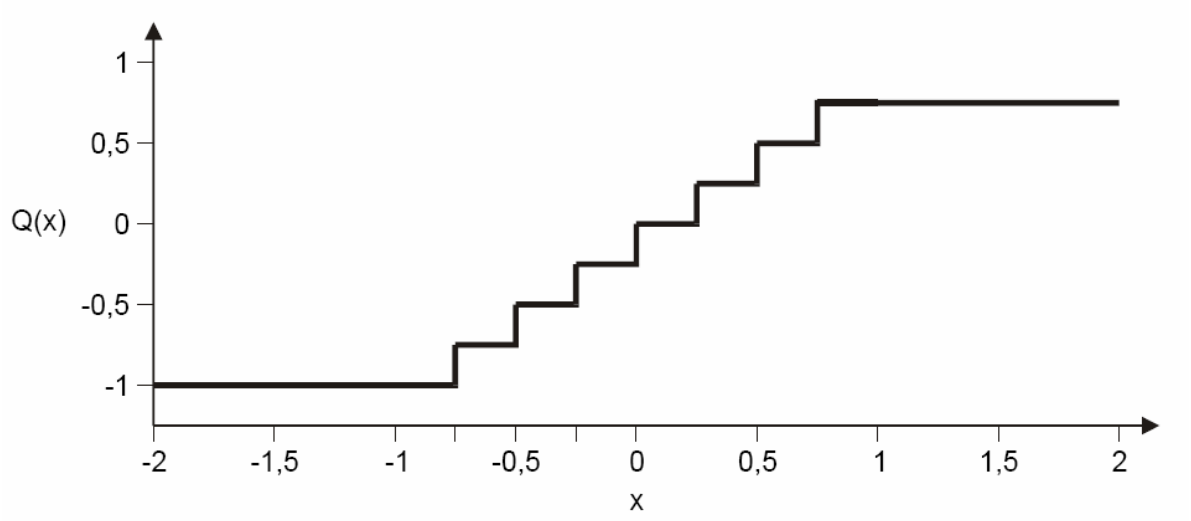

Bild 8.10: Quantisierungskennlinie mit Sättigungsverhalten.

bei einer Addition vor der Berechnung auf dieselbe Bitlänge gebracht. Oder nach einer Multiplikation ist das Ergebnis zu breit und muss verringert werden. Zuerst betrachten wir die Erweiterung auf eine höhere Bitanzahl. Dabei gibt es zwei Möglichkeiten. Zum Einen das so genannte right alignment, bei welchem wie in Kap. 8.1.2.4.3 beschrieben, zusätzliche Bit vor dem MSB mit dem Wert des MSB gefüllt werden. Zum Anderen das left alignment, dabei werden links des LSB Nullen eingefügt. In Bild 8.11 sind beide Varianten bei einer Änderung der Bitbreite von vier auf acht Bit dargestellt. Man erkennt, dass beim left alignment zusätzliche Nachkommastellen hinzugefügt werden, während beim right alignment zusätzliche Vorkommastellen entstehen. Durch Kombination der beiden Verfahren kann jedes I, Q–Format in ein $I+n, Q+m$–Format überführt werden.

Im umgekehrten Fall der Bitreduzierung gibt es ebenfalls zwei Möglichkeiten. Es können die niederwertigsten Nachkommastellen abgeschnitten werden, dies hat allerdings eine höhere Ungenauigkeit zur Folge. Die zweite Möglichkeit besteht darin, dass redundante Vorkommastellen, also Bit mit identischem Inhalt, gestrichen werden. Dabei ist zu Beachten, dass es sich wirklich um redundante Bit handeln muss, da sich sonst der Zahlenwert gravierend ändert (Vorzeichen). Der Vorteil beim zweiten Fall ist, dass die Zahl in derselben Genauigkeit vorliegt, wie vor der Reduzierung. Tabelle 8.3 führt zwei Beispiele der Reduzierung um ein Bit auf. In Beispiel 1 funktioniert das Streichen einer Vorkommastelle und die Zahl bleibt genau erhalten. Im zweiten Beispiel führt dieses Verfahren allerdings zu einem ungültigen Ergebnis. Das Streichen einer Nachkommastelle funktioniert immer, allerdings mit eventuellem Genauigkeitsverlust (Beispiel 1).

8.1.2.5.2 Addition/Subtraktion

Die Addition zweier Festkommazahlen gliedert sich in zwei Schritte. Zuerst müssen die beiden Summanden in das gleiche Format überführt werden. Dies läuft im Falle von $1, Q$–

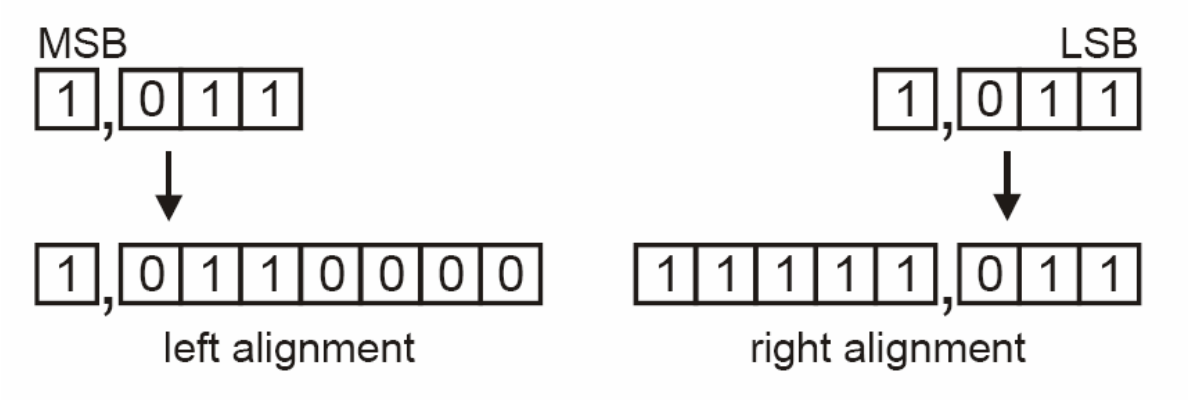

Bild 8.11: Bitbreitenänderung von vier auf acht Bit.

	Beispiel 1		Beispiel 2	
5 Bit Zahl	11,011	$(-0,625)$	10,010	$(-1,75)$
Nachkommastelle abgeschnitten	11,01	$(-0,75)$	10,01	$(-1,75)$
Vorkommastelle gestrichen	1,011	$(-0,625)$	0,010	$(0,375)$

Tabelle 8.3: Reduzierung um ein Bit.

Formaten auf ein left alignment des schmaleren Summanden hinaus. Danach werden sie bitweise addiert. Ein eventueller Übertrag über das MSB hinaus wird ignoriert. Das Ergebnis hat das beschriebene Modulo 2 Überlaufverhalten. Zur sicheren Vermeidung von Überläufen können beide Summanden vor der Addition um eine Vorkommastelle ergänzt werden, das Ergebnis besitzt dann ebenfalls eine Stelle mehr, was oft nicht erwünscht ist. Wie Bild 8.12 zeigt, unterscheidet sich die Festkommaaddition in der Durchführung nicht von der Addition zweier ganzer Zahlen. Eine Subtraktion lässt sich mit Hilfe der Addition durchführen. Es muss nur vor der Addition das Zweierkomplement des Subtrahenden gebildet werden.

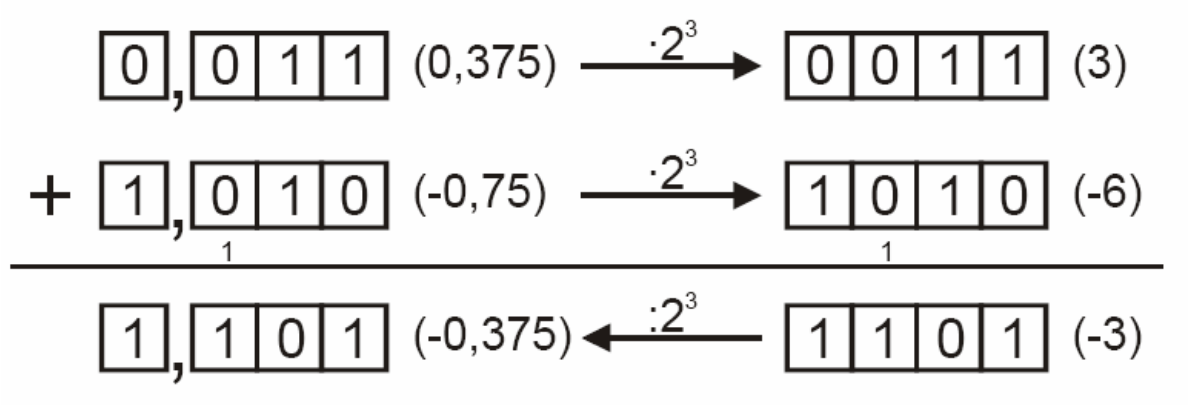

Bild 8.12: Addition von Festkommazahlen

8.1.2.5.3 Multiplikation

Bei der Multiplikation stellt sich die Frage, wie breit das Ergebnis sein muss, damit keine Daten verloren gehen. Zuerst betrachten wir dies bei der Multiplikation zweier ganzer Zahlen der Bitlänge n bzw. m mit $n \geq m$. Die betragsmäßig größtmögliche negative Zahl der beiden Terme ist -2^{n-1} bzw. -2^{m-1}, die größtmögliche positive $2^{n-1}-1$ bzw. $2^{m-1}-1$. In Formel 8.20 wird die maximal mögliche positive Zahl des Ergebnisses bestimmt. Diese ergibt sich entweder aus der Multiplikation der beiden betragsmäßig größten negativen Zahlen oder der beiden größten positiven Zahlen.

$$(2^{n-1}-1)\cdot(2^{m-1}-1) \quad \vee \quad (-2^{n-1})\cdot(-2^{m-1}) \quad \Rightarrow \quad 2^{n+m-2} \tag{8.20}$$

Die Bestimmung des kleinstmöglichen Ergebnisses ist in Formel 8.21 angegeben. Es ergibt sich entweder aus der kleinsten Zahl des 1. Termes (n Bit) mal der größten des 2. (m Bit) oder umgekehrt.

$$\begin{array}{ccc} (2^{n-1}-1)\cdot(-2^{m-1}) & \vee & (-2^{n-1})\cdot(-2^{m-1}-1) \\ -[2^{n+m-2}-2^{m-1}] & \vee & (2^{n+m-2}-2^{n-1}) \\ \Rightarrow & -[2^{n+m-2}-2^{m-1}], & \text{da } 2^{m-1}=2^{n-1} \end{array} \tag{8.21}$$

In Tabelle 8.4 sind die Erkenntnisse nochmals zusammengefasst und die benötigte Bitbreite des Ergebnisses angegeben. Man erkennt, dass die obere Grenze den Bitbedarf von $n+m$ Bit vorgibt. Allerdings lässt sich mit dieser Bitbreite ein wesentlich breiterer Zahlenraum abdecken als vom möglichen Ergebnis gefordert wird. Lediglich der positive Extremwert von 2^{n+m-2} liegt außerhalb des Bereiches einer $n+m-1$ Bit Darstellung. Geht man zu Festkommazahlen im $1,Q$–Format über, so zeigt sich, dass beim Multiplikationsergebnis eine weitere Vorkommastelle entstanden ist (Bild 8.13). Lediglich das Ergebnis der Multiplikation von -1 mit -1 benötigt diese zusätzliche Vorkommastelle, ansonsten ist sie redundant. Bei FIR–Filtern mit normierten Koeffizienten kann dieser

	untere Grenze	obere Grenze
Zahl 1 (n Bit)	-2^{n-1}	$2^{n-1}-1$
Zahl 2 (m Bit)	-2^{m-1}	$2^{m-1}-1$
Ergebnis	$-[2^{n+m-2}-2^{m-1}]$	2^{n+m-2}
$n+m-1$ Bit	-2^{n+m-2}	$2^{n+m-2}-1$
$n+m$ Bit	-2^{n+m-1}	$2^{n+m-1}-1$

Tabelle 8.4: Zahlenbereich bei der Multiplikation.

Extremfall ausgeschlossen und die zweite Vorkommastelle, wie in Kap. 8.1.2.5.1 beschrieben, entfernt werden. Mit dieser Streichung liegt das Ergebnis der Multiplikation zweier $1,Q$–Zahlen als $1,2\cdot Q$–Zahl vor. Bild 8.14 zeigt einen gängigen Multiplizieralgorithmus für Zweierkomplementzahlen, wie er z.B. in dem VHDL–Package IEEE.NUMERIC_STD zum Einsatz kommt. Dabei ist völlig irrelevant, ob es sich um ganze Zahlen oder Festkommazahlen handelt. Bei Kommazahlen muss im Endergebnis nur das Komma an der richtigen Stelle eingefügt werden. Das Resultat hat dann so viele Nachkomastellen, wie

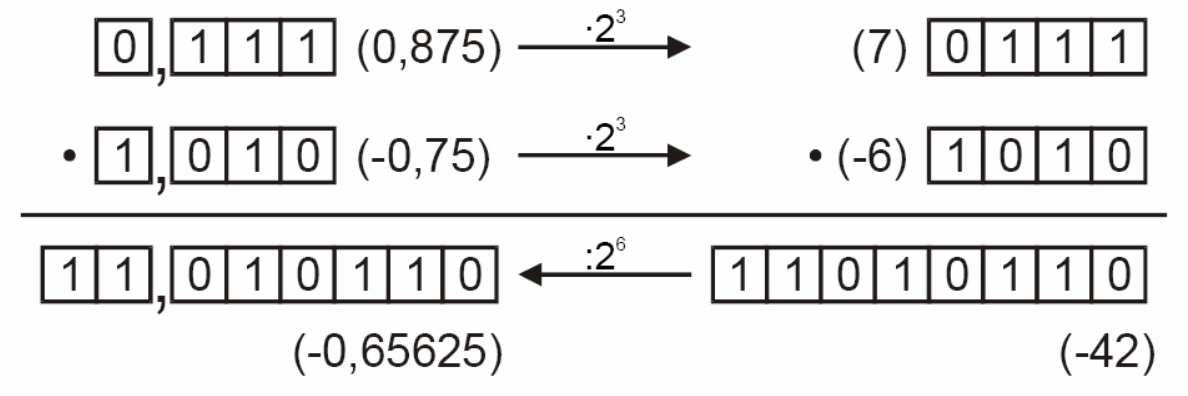

Bild 8.13: Multiplikation von Festkommazahlen.

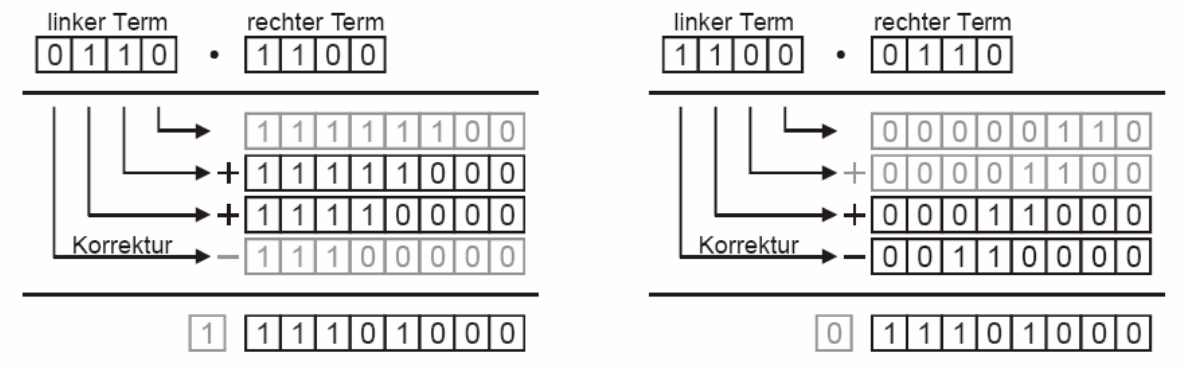

Bild 8.14: Multiplikationsalgorithmus für Zweierkomplementzahlen.

die beiden Terme zusammen. Der Algorithmus gliedert sich in drei Phasen. In der ersten Phase wird der rechte Term mittels right alignment auf die Resultatbreite gebracht. In der zweiten Phase werden die Bit des linken Terms einzeln vom LSB bis zum Bit vor dem MSB abgearbeitet. Ist das jeweilige Bit gesetzt („1“), so wird der erweiterte rechte Term zum Ergebnis addiert, bei gelöschtem Bit wird keine Operation durchgeführt (grau in Bild 8.14). Danach wird der erweiterte rechte Term um eine Stelle nach links geschoben und das nächste Bit betrachtet. Die dritte Phase dient der Korrektur des Ergebnisses, falls der linke Term negativ ist, also sein MSB eine „1“ enthält. In diesem Falle muss der nochmals um eine Stelle verschobene erweiterte rechte Term vom Ergebnis abgezogen werden. Dies geschieht, wie in Kap. 8.1.2.5.2 erwähnt, durch Zweierkomplementbildung und anschließender Addition (in Bild 8.14 nicht dargestellt). Es existieren noch effizientere Algorithmen zur Verwendung in DSPs oder aber auch in programmierbarer Logik, wie z.B. der sogenannte modifizierte Booth–Algorithmus, der den einen Operanden in einem vierstufigen Code codiert und somit etwa auf die halbe Länge reduziert. Welchen Algorithmus der Quartus–Compiler letztendlich anwendet, lies sich leider nicht herausfinden.

8.1.2.6 Gleitkommazahlen

Der Vollständigkeit wegen seien hier die Gleitkommazahlen erwähnt, sie werden im Praktikum allerdings nicht benötigt. Zur Definition sei deshalb auf die Literatur verwiesen. Die Vorteile der Gleitkomma- gegenüber der Festkommadarstellung liegen z.B. in dem größeren darstellbaren Wertebereich und dem kleineren relativen Quantisierungsfehler bei kleinen Zahlen. Nachteilig ist die kompliziertere Arithmetik und die Mehrdeutigkeit vieler Zahlen.

8.1.2.7 Blockgleitkommazahlen

Die Blockgleitkommazahlen versuchen die Vorteile der Gleitkommazahlen und die der Festkommazahlen zu vereinen. Innerhalb eines Rechenblocks werden alle Zahlen mit gleichem Exponenten dargestellt und entsprechen somit Festkommazahlen. Arithmetische Operationen sind einfach zu realisieren. Außerhalb eines solchen Blockes haben die Zahlen dann wieder variable Exponenten. Dies kann beim Koeffizientenspeicher von Filtern hoher Ordnung ausgenutzt werden. Sind z.B. alle Koeffizienten betragsmäßig kleiner als $0,25$ so erhält jeder Koeffizient per Definition einen Exponenten von -2, welcher nicht mit abgespeichert wird, und man kann ohne Genauigkeitsverlust pro Koeffizient zwei Bit einsparen (siehe Tabelle 8.5). Vor dem Multiplizierer müssen diese Bit allerdings wieder eingefügt werden, was allerdings kaum Aufwand bedeutet (siehe Kap. 8.1.2.5.1).

8.1.3 Digitale Filter

Dieses Kapitel behandelt zuerst die allgemeinen Grundlagen digitaler Filter. Anschließend folgt die Aufteilung der Filter in IIR- und FIR–Typen, wobei der Schwerpunkt auf den FIR–Filtern liegt. Die speziellen Eigenschaften dieser Filter werden herausgestellt

Festkommazahl	Blockgleitkommazahl
$0,00101$	$0,101 \cdot 2^{-2}$
$1,11000$	$1,000 \cdot 2^{-2}$

Tabelle 8.5: Biteinsparung durch Blockgleitkommazahlen.

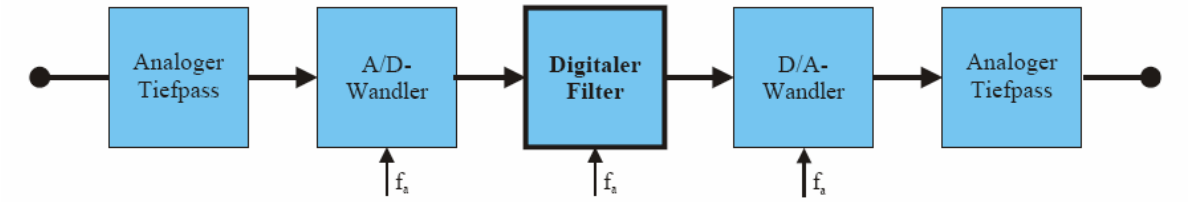

Bild 8.15: Umgebung von digitalen Filtern.

und ihre Umsetzungsmöglichkeiten diskutiert. Darauf folgt eine Betrachtung einfacher FIR–Filter, wie sie im ersten Praktikumsversuch umgesetzt werden sollen. Das Kapitel schließt mit einer Analyse der Designschritte für einen FIR–Filter höherer Ordnung ab.

8.1.3.1 Grundlagen digitale Filter

Der Begriff des Digitalfilters wird in der Literatur unterschiedlich ausgelegt. In vielen Büchern wird er ähnlich den analogen Filtern als *„ein System, das gewisse Frequenzkomponenten im Vergleich zu anderen verändert“* bezeichnet. Es gibt aber auch Definitionen, die den Begriff deutlich weiter fassen und digitale Filter als *„einen auf einem digitalen System implementierten Algorithmus mit irgendwelchen Eigenschaften“* bezeichnen. Übereinstimmend ist in beiden Definitionen, dass es sich bei einem digitalen Filter um ein spezielles digitales System handelt, somit ist das in Kap. 8.1.2.3 erwähnte auf sie anwendbar.

8.1.3.1.1 Umgebung von digitalen Filtern

Digitale Filter werden häufig in einer analogen Umgebung eingesetzt, z.B. bei Audiound Videosignalen. Eine typische Beschaltung ist in Bild 8.15 zu sehen. Der Eingangstiefpass stellt sicher, dass kein Aliasing auftritt, weshalb er auch als Anti–Aliasing-Filter bezeichnet wird. Danach wird das analoge Signal im A/D–Wandler in eine Folge von Abtastwerten umgesetzt. Diese werden im Digitalfilter verarbeitet, was wiederum eine Folge von diskreten Werten ergibt. Der D/A–Wandler setzt das Ergebnis in eine analoge Treppenfunktion um. Der Analoge Tiefpass am Ausgang, auch Rekonstruktionsfilter genannt, glättet schließlich die Kanten der Treppenfunktion und es entsteht wieder ein wert- und zeitkontinuierliches Signal. In vielen Fällen sind die Arbeitsfrequenzen von A/D–Wandler, Digitalfilter und D/A–Wandler identisch, dies ist allerdings kein Zwang. Der digitale Filter kann auf unterschiedlichsten Plattformen realisiert werden. Im Prinzip kann jeder Mikroprozessor als Filter programmiert werden. Besser geeignet sind aber Signalprozessoren, bei denen häufig gebrauchte Funktionen wie z.B. MAC–Operationen

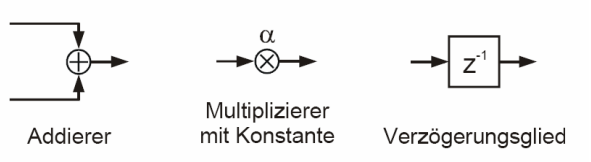

Bild 8.16: Elemente der Digitalfilter.

(**M**ultiply–**AC**cumulate – jeweils zwei Werte multiplizieren und die Ergebnisse aufaddieren) besonders schnell implementiert sind. Es existieren auch spezielle Digitalfilterchips für besonders hohe Abtastfrequenzen oder hohe Stückzahlen. Im Praktikum werden die Filter auf einem programmierbaren Logikbaustein realisiert.

8.1.3.2 Elemente der Digitalfilter

Wie bereits erwähnt, ist ein digitaler Filter ein digitales System. Leicht einzusehen ist, dass es sich sogar um ein LTD–System handelt. Betrachtet man die Differenzengleichung eines solchen Systems (Formel 8.10), so zeigt sich, dass zur Realisierung lediglich drei Elemente benötigt werden. Dies sind Addierer, Multiplizierer mit einer Konstanten und Verzögerungsglieder. Bild 8.16 zeigt die grafische Darstellung dieser Elemente, wie sie im Folgenden verwendet werden. Das Verzögerungsglied entspricht in der Schaltungsumsetzung einem D–Flipflop. Die Bezeichnung des Verzögerungsgliedes mit z^{-1} deutet an, dass wir uns bei der Systembeschreibung als Blockschaltbild im Frequenzbereich bewegen. Dies ist allerdings eine willkürliche Festlegung. Sie erscheint sinnvoll, da man von den analogen Filtern gewohnt ist, den Amplituden- und Phasengang anzugeben, und dies ist über die Systemfunktion auf einfachem Wege möglich.

8.1.3.2.1 Systemfunktion

Es stellt sich die Frage, wie man aus einem Blockschaltbild die Systemfunktion oder umgekehrt erhält. Dazu betrachten wir das relativ einfache System in Bild 8.17. Aus dem Blockschaltbild lässt sich direkt die Differenzengleichung ablesen:

$$y(n) = \alpha_0 x(n) + \alpha_1 x(n-1) - \beta_1 y(n-1) \tag{8.22}$$

Diese überführen wir mittels Z–Transformation in den Frequenzbereich und erhalten:

$$\begin{gathered} Y(z) = \alpha_0 X(z) + \alpha_1 z^{-1} X(z) - \beta_1 z^{-1} Y(z) \\ Y(z)[1 + \beta_1 z^{-1}] = X(z)[\alpha_0 + \alpha_1 z^{-1}] \end{gathered} \tag{8.23}$$

Den Weg über die Differenzengleichung hätte man sich sparen und Formel 8.23 direkt aus dem Blockschaltbild entnehmen können. So hat sich jedoch nochmals gezeigt, dass beide Formen äquivalent sind. Nach einer weiteren Umstellung erhalten wir die Systemfunktion:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\alpha_0 + \alpha_1 z^{-1}}{1 + \beta_1 z^{-1}} \tag{8.24}$$

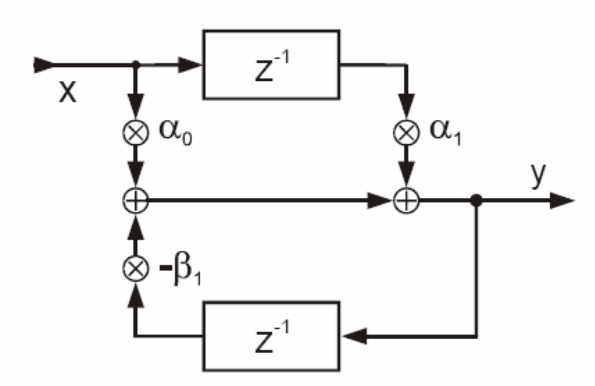

Bild 8.17: Systembeispiel.

Diese hat, da es sich um ein LTD–System handelt, die Gestalt aus Formel 8.13. Der Zusammenhang zwischen den Koeffizienten des Blockschaltbilds und denen der Formel 8.24 lässt sich leicht erkennen. Somit kann man einfach aus dem Blockschaltbild die Systemfunktion herauslesen. Die Impulsantwort lässt sich durch inverse Z–Transformation bestimmen.

8.1.3.2.2 Frequenzgang

Unter dem Frequenzgang eines Filters versteht man im Allgemeinen den komplexen Frequenzgang, der sich aus der Fouriertransformation der Impulsantwort ergibt, und somit identisch mit der Übertragungsfunktion ist. Wie bereits erwähnt, lässt er sich aber auch direkt aus der Systemfunktion durch Variablentransformation bestimmen (Kap. 8.1.2.3.3). Aus diesem Frequenzgang lassen sich der Amplituden- und Phasengang des Filters bestimmen. Die Amplitude (A) ist der Betrag des Frequenzganges und die Phase (φ) der Winkel:

$$H(\omega) = A(\omega) \cdot e^{-j\varphi()\omega} \tag{8.25}$$

8.1.3.2.3 Anordnung der Elemente

Es gibt verschiedene Wege, die drei Grundelemente, aus denen sich digitale Filter zusammensetzen, anzuordnen. Bei den hier behandelten drei Versionen ändern sich die Filtereigenschaften durch Wechsel auf eine andere Anordnung nicht, jedoch der Ressourcenverbrauch und Rechenaufwand bei der Realisierung. Bild 8.18 zeigt eine mögliche Version mit vier Stufen, die wir schon aus Bild 8.17 in einstufiger Version kennen. Sie besitzt einen globalen Summierer am Ausgang und benötigt für jede Stufe zwei Verzögerungsglieder. Fasst man die zwei Verzögerungsglieder pro Stufe zu einem zusammen, so gelangt man zur Struktur der Bild 8.19. Für die Verwendung im Digitalrechner oder – wie in den Praktikumsversuchen gezeigt wird – mit programmierbaren Logikbausteinen gibt es eine weitere äquivalente Anordnung. Diese ist in Bild 8.20 wiedergegeben und zeichnet sich durch verteilte Summierer aus.

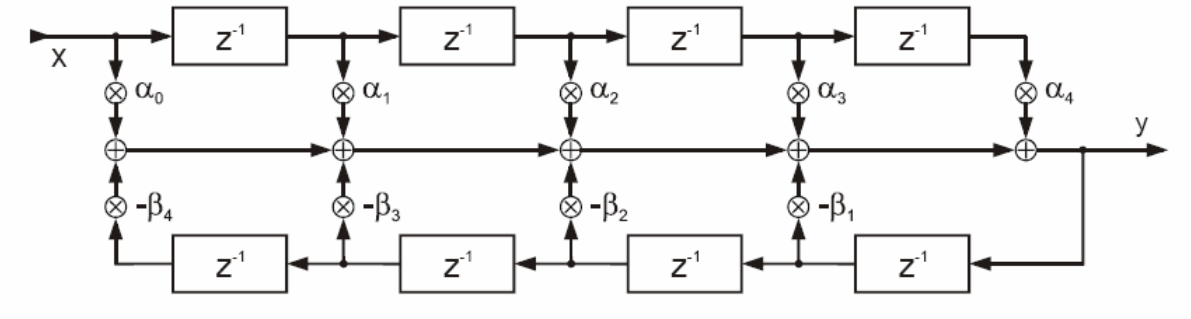

Bild 8.18: Elementanordnung mit globalem Summierer am Ausgang.

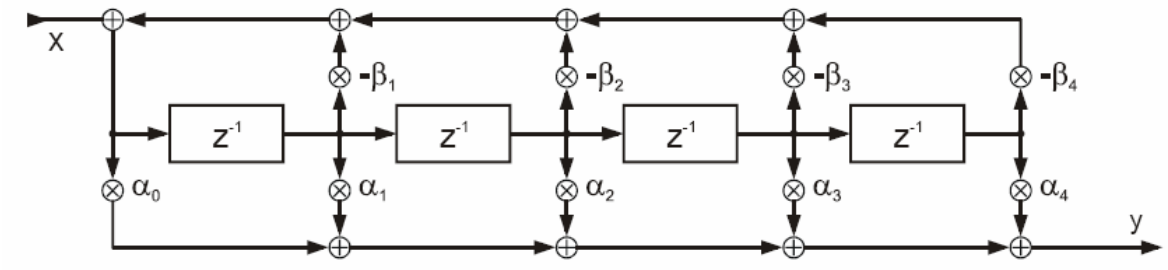

Bild 8.19: Elementanordnung mit je einem globalen Summierer am Ein- und Ausgang.

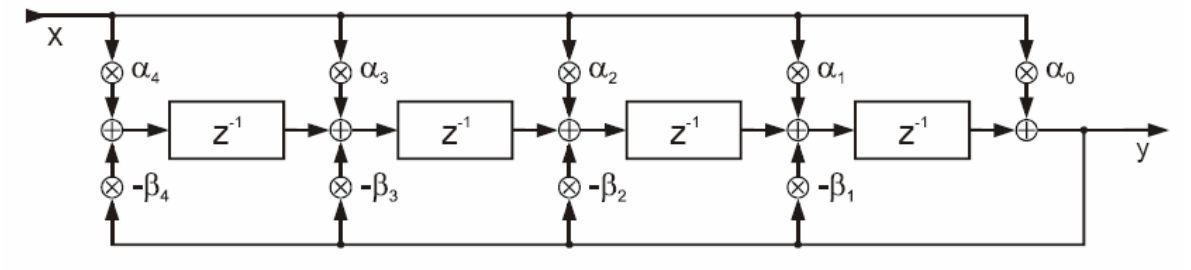

Bild 8.20: Elementanordnung mit verteilten Summierern.

8.1.3.2.4 Filterordnung

Bei digitalen Filtern bezeichnet man als Ordnung die Anzahl der Stufen, die der Filter besitzt. Diese ist identisch mit der Anzahl der Verzögerungselemente wenn man eine Anordnung nach Bild 8.19 oder Bild 8.20 wählt. Es kann keinerlei Aussage über den Amplitudengang des Filters anhand seiner Ordnung getroffen werden. Im folgenden wird die Ordnung eines Filters mit dem Buchstaben N bezeichnet.

8.1.3.3 Stichwort Echtzeitsystem

Die meisten digitalen Filter werden als Echtzeitsysteme eingesetzt. Was bedeutet aber Echtzeit? Es bedeutet, dass das System die Eingangsdaten mit der anfallenden Rate verarbeiten können muss. Es ist einleuchtend, dass ein realer digitaler Filter eine gewisse Zeit benötigt, um für einen neuen Eingangswert den Ausgangswert zu bestimmen, dieser also nicht sofort zur Verfügung steht. Durch Pipeline–Techniken kann dieser sogar erst mehrere Eingangstakte später endgültig bestimmt sein.

8.1.3.4 Pipelining

Das Pipelining–Prinzip ist aus modernem Prozessordesign nicht mehr wegzudenken. Nur dadurch ist die Abarbeitung eines komplexen Befehls in einem Taktzyklus möglich. Wir wollen das Prinzip an einem FIR–Filter zweiter Ordnung in paralleler Ausführung betrachten. Aus Kap. 8.1.2.5.3 wird ersichtlich, dass in Bezug auf die Arbeitsgeschwindigkeit die Multiplizierer die kritischen Elemente sind. Der nachfolgende Addierer kann seine Berechnung erst beginnen, wenn die Multiplizierer ihre Operationen abgeschlossen haben. Durch Einführen einer Pipeline lässt sich die Multiplikation in den einen Systemtakt und die Addition in den folgenden Takt verschieben. Dazu ist lediglich ein Register pro Multiplizierer notwendig, wie Bild 8.21 veranschaulicht. Ist die Pipeline mit Werten gefüllt, was in diesem Falle nach einem Systemtakt der Fall ist, so wird pro Taktzyklus ein Wert berechnet. Das Ausgangssignal ist allerdings um einen Takt verzögert, was aber in den meisten Anwendungen keine Probleme bereitet.

Typische Addierer können nur zwei Eingangswerte verarbeiten, weshalb der in Bild 8.21 verwendete Typ mit drei Eingängen bei der Realisierung in zwei Addierer aufgespaltet wird. Dies könnte wiederum zu einer Herabsetzung der möglichen Arbeitsgeschwindigkeit führen. Umgehen lässt sich dies durch Einfügen einer weiteren Pipelinestufe. Man erhält den Aufbau nach Bild 8.22. Diese Pipeline ist jetzt allerdings erst nach zwei Systemtakten gefüllt und somit das Ausgangssignal entsprechend verzögert. Verschiedene Möglichkeiten von Pipelining und Parallelverarbeitung werden in der Literatur mit Bezug auf digitale Filter genau beschrieben.

8.1.3.5 FIR–Filter

FIR–Filter bedeutet Finite Impuls Response Filter, er bezeichnet somit einen Filter mit endlicher Impulsantwort. Der Ausgang hängt nur vom aktuellen Eingangswert und

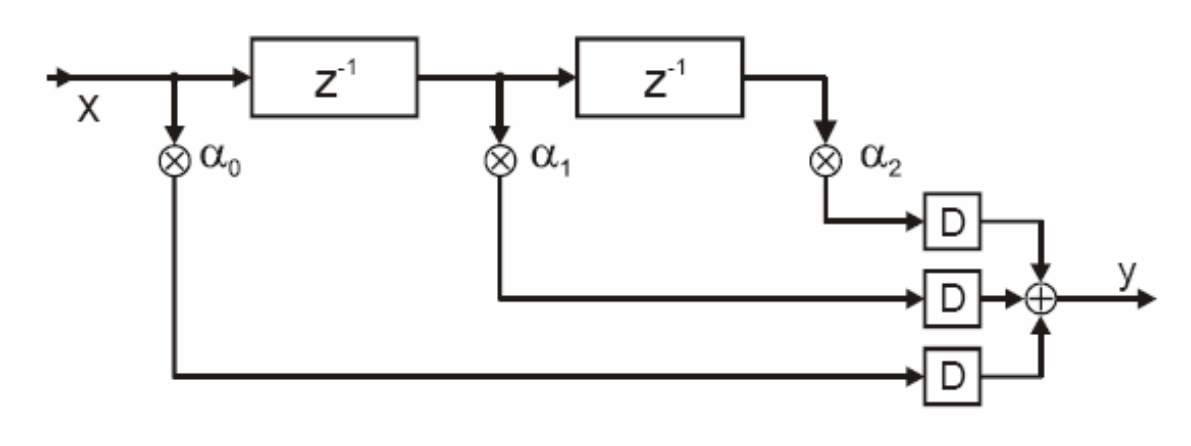

Bild 8.21: FIR–Filter mit Pipelining.

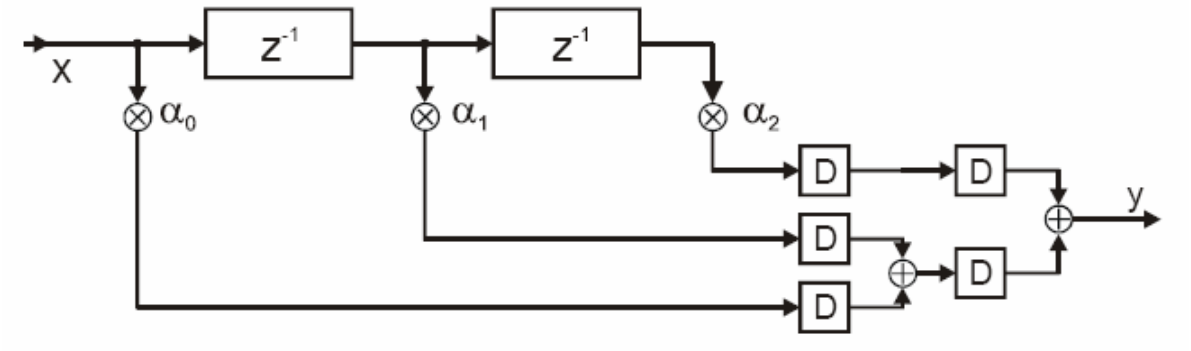

Bild 8.22: Erweiterte Pipelinestruktur.

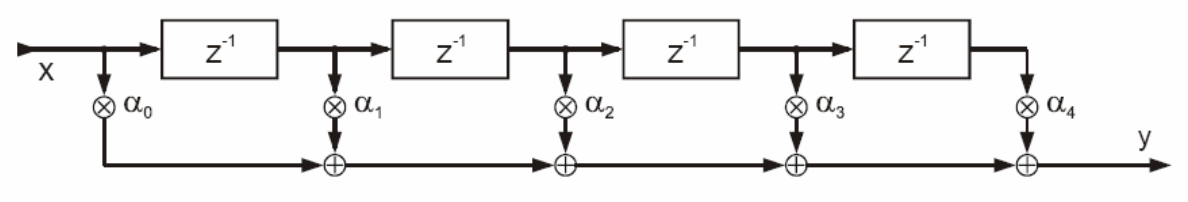

Bild 8.23: FIR–Filter mit globalem Summierer.

von endlich vielen vergangenen Eingangswerten ab und besitzt somit keine Rückkopplung. Solche Systeme werden auch als nichtrekursive Systeme bezeichnet. Die Differenzengleichung und Systemfunktion vereinfachen sich zu:

$$y[n] = \sum_{i=0}^{N} \alpha_i \cdot x[n-i] \tag{8.26}$$

$$H(z = \sum_{i=0}^{N} \alpha_i \cdot z^{-i} = K \cdot \frac{\prod_{i=1}^{N}(z - n_i)}{z^N} \tag{8.27}$$

Die Systemfunktion besitzt nur Pole im Nullpunkt. Das System ist folglich für beliebige Konstanten stabil. Eine Stabilitätsüberprüfung kann somit für FIR–Filter komplett entfallen. Bei den Elementanordnungen fallen die beiden Versionen mit globalem Summierer zu einer zusammen, das Ergebnis zeigt Bild 8.23. Somit stimmen beim FIR–Filter die Ordnung (N) und die Anzahl der Verzögerungsglieder überein. Die Anzahl der Koeffizienten beträgt $N + 1$.

8.1.3.5.1 Eigenschaften der FIR–Filter

Eine sehr interessante Eigenschaft eines FIR–Filters ist, dass er auf den diskreten Einheitsimpuls mit der Folge seiner Koeffizienten antwortet. Dieses Verhalten lässt sich aus der inversen Z–Transformation der Systemfunktion von Formel 8.6 ablesen, was ja gerade die Impulsantwort des Systems ergibt:

$$h(n) = \sum_{i=0}^{N} \alpha_i \cdot \delta(n-i) \tag{8.28}$$

Dies ist besonders für die Simulation der Filter vorteilhaft, da der diskrete Einheitsimpuls ein leicht zu realisierender Stimulus ist. Des weiteren ist die Gleichspannungsverstärkung eines FIR–Filters gleich der Summe seiner Koeffizienten. Aus der Differenzengleichung von Formel 8.26 folgt für ein konstantes Eingangssignal $x(n) = k$ (Gleichspannung) das konstante Ausgangssignal:

$$y[n] = k \cdot \sum_{i=0}^{N} \alpha_i \tag{8.29}$$

Dies ist eine um die Summe aller Koeffizienten verstärkte Gleichspannung. Multipliziert man alle Koeffizienten eines FIR–Filters mit derselben Konstanten, so ändert sich nur dessen Verstärkung, nicht aber seine Charakteristik. Diese Eigenschaft lässt sich an der Systemfunkti on aus Formel 8.25 am einfachsten ablesen. Werden alle Koeffizienten mit der Konstanten c multipliziert, so ändert sich die Systemfunktion zu:

$$H(z = c \cdot \sum_{i=0}^{N} \alpha_i \cdot z^{-i} = c \cdot K \cdot \frac{\prod_{i=1}^{N}(z - n_i)}{z^N} \tag{8.30}$$

Die Lage der Pol- und Nullstellen bleibt erhalten, und somit auch die Filtercharakteristik. Eine sehr interessante Eigenschaft von FIR–Filtern ergibt sich, wenn man sich auf symmetrische Koeffizienten beschränkt, entweder in gerader oder ungerader Symmetrie. Dann ergibt sich daraus ein linearer Phasenverlauf des Filters von:

$$\varphi(f) = \begin{cases} -\pi \cdot N \cdot \frac{f}{f_a} & \text{für gerade Symmetrie} \\ -\pi \cdot N \cdot \frac{f}{f_a} + \frac{\pi}{2} & \text{für ungerade Symmetrie} \end{cases} \tag{8.31}$$

Aus dem Phasenverlauf ergibt sich die Gruppenlaufzeit zu:

$$t_q = -\frac{\partial \varphi}{\partial \omega} = -\frac{\partial \varphi}{\partial f} \cdot \frac{\partial f}{\partial \omega} = -\frac{\partial \varphi}{\partial f} \cdot (-\frac{1}{2\pi}) \tag{8.32}$$

$$t_q = \frac{1}{2} \cdot \frac{N}{f_a} = \frac{1}{2} \cdot N \cdot T_a \tag{8.33}$$

Dies ist für beliebige symmetrische Koeffizienten der Fall und somit völlig unabhängig von der Filtercharakteristik. Die lineare Phase ist eine wünschenswerte Eigenschaft von Filtern, deshalb werden hauptsächlich FIR–Filter mit symmetrischen Koeffizienten verwendet. In diesem Praktikum werden wir nur solche Filter realisieren. Ist ein linearer Phasenverlauf nicht vonnöten, so greift man besser auf ein IIR–Filter zurück, welcher weniger Ressourcen beansprucht.

8.1.3.6 IIR-Filter

Das allgemeine LTD–System mit der Systemfunktion nach Formel 8.16 besitzt eine theoretisch unendlich lange Impulsantwort. Es wird deshalb auch als Infinite Impuls Response (IIR) Filter bezeichnet. Das einfache Beispiel in Bild reffig:B-25 zeigt das Prinzip eines IIR–Filters. Er besitzt immer eine Rückkopplung, was der Grund für die theoretisch unendlich lange Impulsantwort ist. Bei einer praktischen Umsetzung besitzt die Impulsantwort des IIR–Filters nach Bild 8.24 allerdings eine endliche Impulsantwort, da irgendwann die Quantisierungsgrenze des Systems erreicht ist.

8.1.3.6.1 Vergleich mit IIR–Filternn

FIR–Filter benötigen im Vergleich zu IIR–Filtern eine weitaus höhere Ordnung zur Umsetzung derselben Grenzfrequenz. So benötigt ein Tiefpass mit $f_g = 0.01 \cdot f_a$ mindestens die Ordnung 65, als IIR–Filter lässt er sich schon in erster Ordnung realisieren.

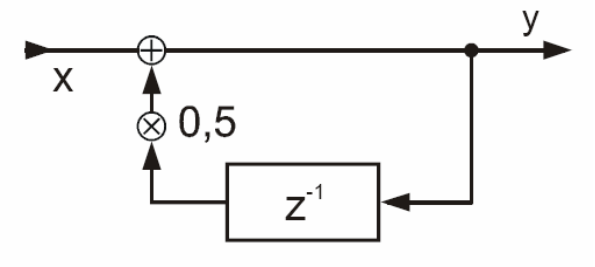

Bild 8.24: IIR–Filter.

Selbst wenn man die Selektivität auf die Anzahl der nötigen MAC–Operationen bezieht, kommen die FIR–Filter nicht gut weg, obwohl sie pro Filterordnung nur eine MAC–Operation benötigen, IIR–Filter hingegen zwei. Der Hauptvorteil der FIR–Filter liegt in der einfachen Realisierbarkeit einer linearen Phase und der garantierten Stabilität. In Tabelle 8.6 findet sich eine Gegenüberstellung der beiden Filtertypen.

Filtermerkmal	FIR–Filter	IIR–Filter
Selektivität	gering	**hoch**
erforderliche Ordnung	hoch	**niedrig**
Anzahl MAC–Operationen	viele	**wenige**
Speicherbedarf	hoch	**gering**
lineare Phase	**problemlos**	kaum möglich
konstante Gruppenlaufzeit	**problemlos**	kaum möglich
Stabilität	**unbedingt**	bedingt
erforderliche Wortbreite	**mäßig**	hoch
erforderliche Koeffizientengenauigkeit	**mäßig**	hoch
Grenzzyklen	**keine**	vorhanden
adaptives Filter	**möglich**	kaum möglich

Tabelle 8.6: Gegenüberstellung von FIR- und IIR–Filtern.

8.1.3.6.2 Parallele Realisierung

Parallele Realisierung ist nur mit programmierbarer Logik oder speziellen DSPs möglich. Über die Programmierung eines Mikroprozessors lässt sich keine parallele Struktur umsetzen, da die Software sequentiell abgearbeitet wird. Parallele Realisierung bedeutet, dass jeder Multiplizierer wirklich als eigene Einheit umgesetzt wird. Somit benötigt ein FIR–Filter vierter Ordnung fünf Multiplizierer (vgl. Bild 8.23). Dies bedeutet einen mit der Filterordnung wachsenden Hardwareaufwand, der meist nur für kleine Ordnungen sinnvoll ist. Der entscheidende Vorteil einer parallelen Struktur ist die Verarbeitungsgeschwindigkeit. Mit jedem Systemtakt wird ein neuer Ausgangswert berechnet. Die Arbeitsgeschwindigkeit der Multiplizierer geben die maximal zu verarbeitende Abtastrate an. Die Geschwindigkeit der Addierer ist bis jetzt vernachlässigt worden. Im Allgemeinen sind sie schneller als die Multiplizierer. Und dank Pipelining (Kap. 8.1.3.3)

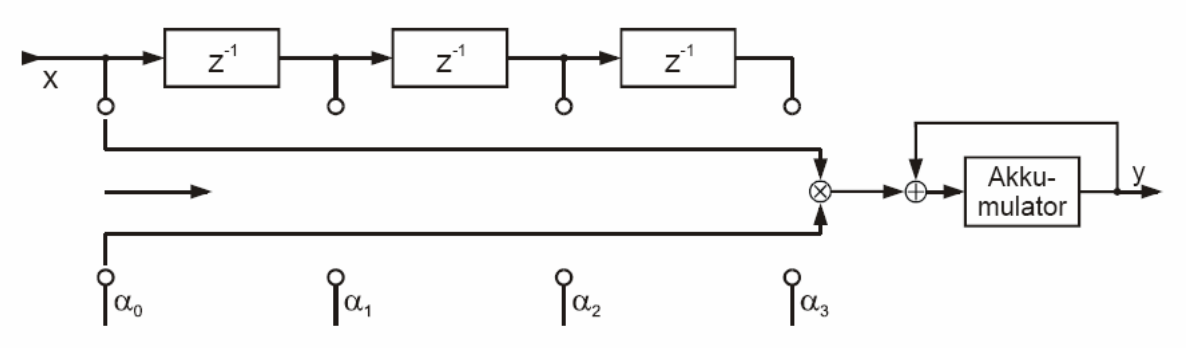

Bild 8.25: Serielle Realisierung eines FIR–Filters.

kann die Multiplikation während eines anderen Systemtakts erfolgen als die Addition. Daher war die Annahme des Multiplizierers als begrenzendes Element gerechtfertigt.

8.1.3.6.3 Serielle Realisierung

Bei Filtern hoher Ordnung oder zur Ressourcenschonung kann man den Filter auch seriell implementieren. Dabei geht man von der Anordnung mit einem globalen Summierer (Bild 8.23) aus und ersetzt die Multiplizierer durch einen einzigen, der nacheinander mit den Eingangswerten und den entsprechenden Koeffizienten gespeist wird. Ein Akkumulator am Ausgang dient als globaler Summierer und addiert die einzelnen Multiplikationsergebnisse auf. Die Anordnung ist in Bild 8.25 zu sehen. Dabei ist, wie bei Mikroprozessoren üblich, lediglich das Summierregister als Akkumulator bezeichnet. Umgesetzt wird der serielle Filter zumeist nach Bild 8.26 mit zwei Ringspeichern. So müssen nicht die kompletten Speicherinhalte geschoben werden, sondern ein Zeiger durchläuft alle Speicherstellen. Beim Wertespeicher zeigt ein zweiter Zeiger auf den jeweils ältesten Wert, der mit dem nächsten ankommenden Wert überschrieben wird. Der Programmieraufwand ist gegenüber der parallelen Realisierung deutlich gestiegen, was sich in Umsetzung einer solchen seriellen Filterstruktur niederschlägt. Eine solche Struktur benötigt zwei Takte, zum Einen den Takt der Eingangssignale und zum Anderen einen schnelleren Systemtakt, mit dem die MAC–Operationen getaktet werden. Bei einem Filter der Ordnung N muss der Systemtakt mindestens $N + 1$ mal so schnell wie der Eingangstakt sein. Der Systemtakt hat aber keinerlei Einfluss auf die Gruppenlaufzeit, die nur von der Filterordnung abhängt (Formel 8.33).

8.1.3.6.4 Gemischte Realisierung

Reicht die Arbeitsgeschwindigkeit einer seriellen Realisierung nicht aus, so lässt sich der Filter teilweise parallel umsetzen. Verwendet man beispielsweise zwei Multiplizierer, so reduziert sich der nötige Systemtakt schon auf $(N+1)/2$. Durch Hinzufügen weiterer Multiplizierer lässt sich dieser Wert weiter verringern. Allerdings muss der Akkumulator für jeden zusätzlichen Multiplizierer einen zusätzlichen Eingang bereitstellen.

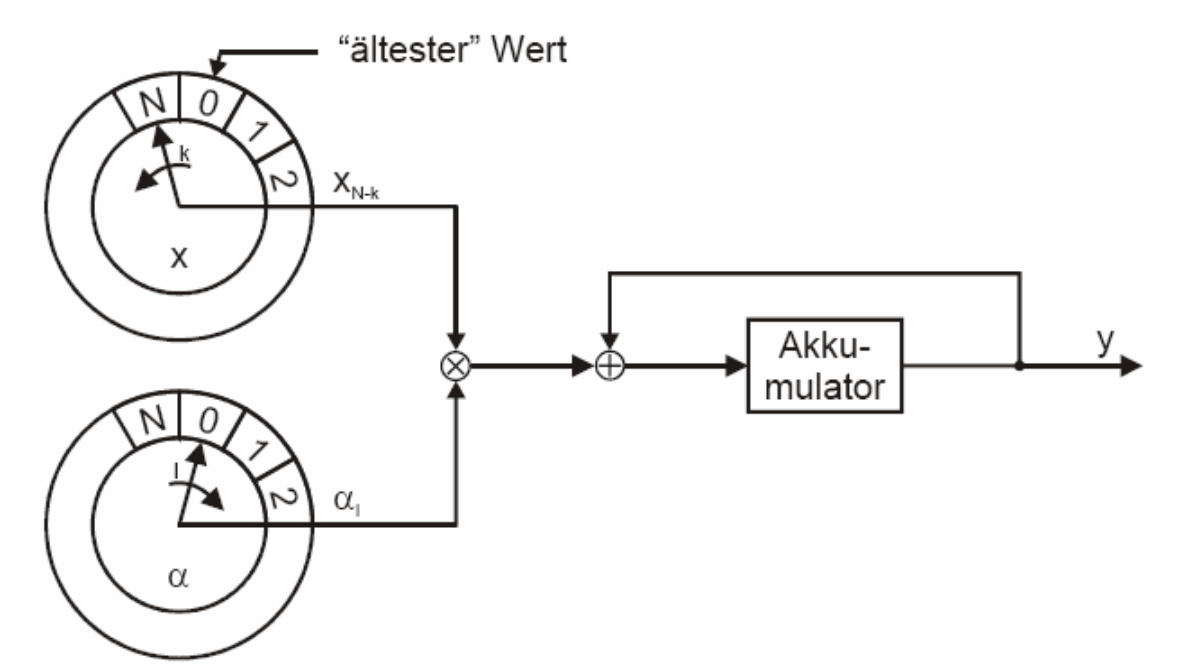

Bild 8.26: Praktische Umsetzung des seriellen Filters.

8.1.3.7 Beispiele einfacher FIR–Filter

Zum besseren Verständnis sollen einige Beispiele von einfachen Filtern mit ihren jeweiligen Charakteristika aufgezeigt werden. Besonders die Filter zweiter Ordnung werden genauer beleuchtet, da diese im ersten Versuch umgesetzt werden.

	erste Möglichkeit	zweite Möglichkeit	dritte Möglichkeit
α_0	$0,5$	$0,5$	$-0,5$
α_1	$0,5$	$-0,5$	$0,5$
Filtertyp	Tiefpass	Hochpass	Hochpass

Tabelle 8.7: Mögliche Koeffizienten für FIR–Filter erster Ordnung.

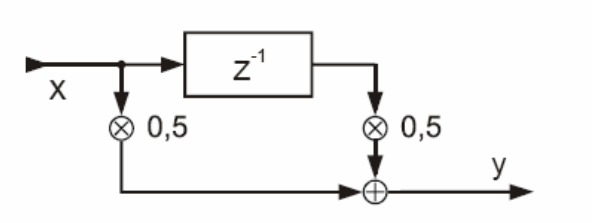

Bild 8.27: FIR–Tiefpass erster Ordnung.

8.1.3.7.1 FIR–Filter erster Ordnung

Ein FIR–Filter erster Ordnung besteht aus nur einer Filterstufe und benötigt somit ein Verzögerungselement, zwei Multiplizierer und einen Addierer. Da wir uns auf Filter mit symmetrischen Koeffizienten beschränken und die Betragssumme aller Koeffizienten auf eins normiert sein soll, ergeben sich die drei Möglichkeiten nach Tabelle 8.7, wobei die letzten Beiden von den Filtereigenschaften her identisch sind. Andere Filtercharakteristiken sind mit einem FIR–Filter erster Ordnung nicht zu realisieren.

8.1.3.7.2 Tiefpass erster Ordnung

Ein einfaches Beispiel für einen FIR–Filter stellt der Tiefpass erster Ordnung nach Bild 8.27 dar. Dass es sich tatsächlich um einen Tiefpass handelt, kann man sich an seiner Gleichspannungsverstärkung von eins ($x[n] \equiv 1 \Rightarrow y[n] \equiv 1$) und an seiner Unterdrückung eines Signals der halben Abtastfrequenz ($x[n] = [1; -1; 1; -1; ...] \Rightarrow y[n] \equiv 0$) verdeutlichen. Der Amplitudengang lässt sich noch relativ leicht analytisch ableiten, wie dies in der Literatur gezeigt wird. Den Amplitudengang zeigt Bild 8.28 einmal über der normierten Frequenz und einmal doppeltlogarithmisch aufgetragen. Das Auftragen über der normierten Frequenz ($F = f/f_a$) hat den Vorteil, dass es unabhängig von der Abtastfrequenz die Filtereigenschaften verdeutlicht. Die 3 dB Grenzfrequenz beträgt 1/2 der Abtastfrequenz. In Bild 8.28 ist die 3 dB Grenze als Linie mit eingezeichnet und man erkennt, dass der Amplitudengang diese bei 10 MHz schneidet.

8.1.3.8 Hochpass erster Ordnung

Einen Hochpass erhält man, indem man die Koeffizienten $0,5$ und $-0,5$ verwendet. Stellt man die selben Überlegungen wie für den Tiefpass an, so erkennt man eine Gleichstromverstärkung von Null ($x[n] \equiv 1 \Rightarrow y[n] \equiv 0$) und eine Verstärkung bei der halben Abtastfrequenz von eins ($x[n] = [1; -1; 1; -1; ...] \Rightarrow y[n] \equiv 1$). Es handelt sich somit tatsächlich um einen Hochpass.

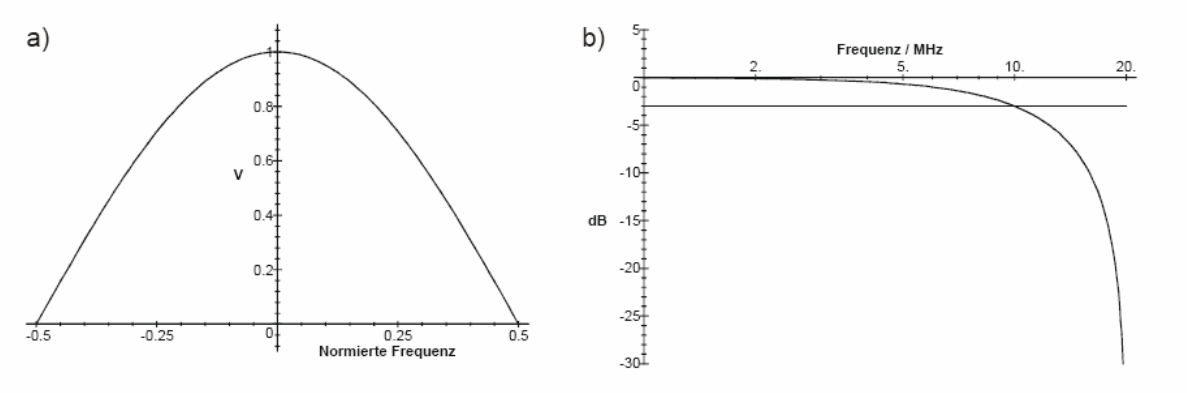

Bild 8.28: Amplitudengang des FIR–Tiefpasses erster Ordnung über der normierten Frequenz (a) und doppeltlogarithmischer Abtastfrequenz $f_a = 40$ MHz (b).

	Tiefpass	Hochpass	Bandpass	Bandsperre
α_0	$0,25$	$0,25$	$-0,5$	$0,5$
α_1	$0,5$	$-0,5$	0	0
α_2	$0,25$	$0,25$	$0,5$	$0,5$

Tabelle 8.8: Betrachtete Koeffizienten für FIR–Filter zweiter Ordnung.

8.1.3.9 FIR–Filter zweiter Ordnung

Ein FIR–Filter zweiter Ordnung besitzt drei Koeffizienten und somit schon deutlich mehr Variationsmöglichkeiten als ein Filter erster Ordnung. In den folgenden Unterkapiteln werden wir die Varianten nach Tabelle 8.8 näher untersuchen. Diese sollen im ersten Filterversuch auf dem programmierbaren Logikbaustein implementiert werden.

8.1.3.9.1 Tiefpass zweiter Ordnung

In Bild 8.29 ist ein Tiefpass zweiter Ordnung im Blockschaltbild mit verteilten Summierern dargestellt. Den Überlegungen von Kap. 8.1.3.6.1 bezüglich Gleichspannungsverstärkung und Verstärkung bei $f_a/2$ fügen wir noch die Überlegung für $f_a/4$ hinzu, was einer Eingangsfolge von $x[n] = [1; 0; -1; 0; 1; ...]$ entspricht. In Tabelle 8.9 sind die Ergebnisse tabelliert. Diese drei betrachteten Frequenzen lassen sich leicht als Stimuli zur Filtersimulation nutzen, wovon bei der Versuchsdurchführung Gebrauch gemacht wird. Der Amplitudengang des Filters ist in Bild 8.30 zu sehen. Zum Vergleich ist nochmals

Frequenz	0	$\frac{f_a}{4}$	$\frac{f_a}{2}$
$x[n]$	$\equiv 1$	$1; 0; -1; 0; 1; ...$	$1; -1; 1; -1; ...$
$y[n]$	$\equiv 1$	$0; -\frac{1}{2}; 0; \frac{1}{2}; 0; ...$	$\equiv 1$
V	1	$\frac{1}{2}$	0

Tabelle 8.9: Überlegungen zum Tiefpass zweiter Ordnung.

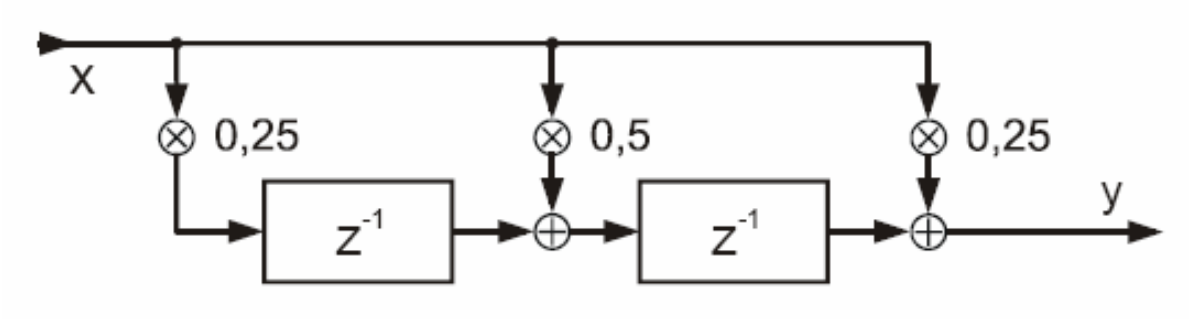

Bild 8.29: FIR–Tiefpass zweiter Ordnung.

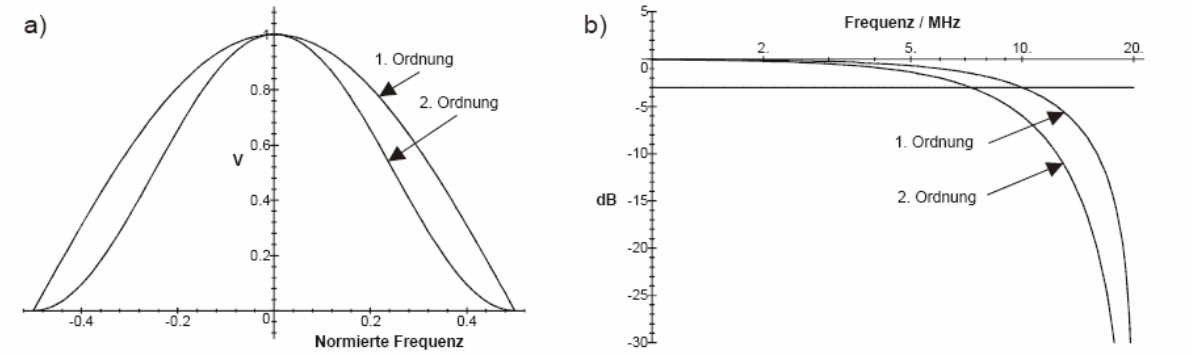

Bild 8.30: Amplitudengang des FIR–Tiefpasses erster und zweiter Ordnung über der normierten Frequenz (a) und doppeltlogarithmischer Abtastfrequenz f_a = 40 MHz (b).

der Amplitudenverlauf des Tiefpassfilters erster Ordnung mit eingezeichnet. Die Grenzfrequenz beträgt ca. das $0,182$–fache der Abtastfrequenz (ca. $7,27$ MHz bei 40 MHz Abtastfrequenz). Sie ist unabhängig von der Grenzfrequenz des Filters erster Ordnung, da die Filterkoeffizienten vorgegeben sind und nicht für eine bestimmte Grenzfrequenz berechnet wurden. Eine Übereinstimmung ist somit erst mit FIR–Filtern höherer Ordnung möglich. Als „theoretischer" Beweis der Funktionalität wurde der Filter durch diskrete Sinusschwingungen angeregt und die Ausgabe aufgezeichnet. Die berechneten Werte sind in Bild 8.31 grafisch dargestellt. Sehr gut zu erkennen ist die konstante Gruppenlaufzeit von einem Abtastintervall, wie es nach Formel 8.12 zu erwarten war. Im rechten Teil ist die Abschwächung des Eingangssignals mit der Frequenz $1/2\ f_a$ auf den halben Wert zu sehen.

8.1.3.10 Hochpass zweiter Ordnung

Einen Hochpass zweiter Ordnung erhält man, indem man in das Blockschaltbild nach Bild 8.30 die Koeffizienten für einen Hochpass von Tabelle 8.10 einsetzt. Überprüft man das Verhalten für die drei Frequenzen 0, $f_a/4$ und $f_a/2$, so bestätigt sich das Hochpassverhalten. Die Ergebnisse sind in Tabelle 8.10 protokolliert. Bild 8.32 zeigt den Ampli-

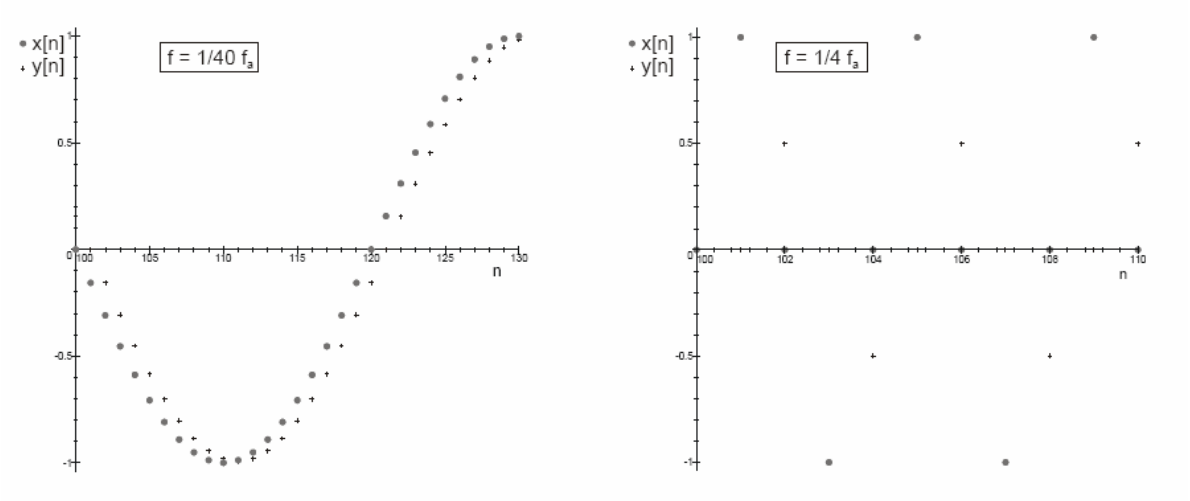

Bild 8.31: Anregungsbeispiele des Tiefpasses zweiter Ordnung.

Frequenz	0	$\frac{f_a}{4}$	$\frac{f_a}{2}$
$x[n]$	$\equiv 1$	$1; 0; -1; 0; 1; ...$	$1; -1; 1; -1; ...$
$y[n]$	$\equiv 0$	$0; \frac{1}{2}; 0; -\frac{1}{2}; 0; ...$	$1; -1; 1; -1; ...$

Tabelle 8.10: Überlegungen zum Hochpass zweiter Ordnung.

tudengang des FIR–Hochpasses erster und zweiter Ordnung. Die Grenzfrequenz beim Filter zweiter Ordnung liegt beim ca. $0,318$–fachem der Abtastfrequenz (ca. $12,73$ MHz bei $f_a = 40$ MHz).

8.1.3.11 Bandpass und Bandsperre zweiter Ordnung

Bei den Koeffizienten für einen Bandpass bzw. Bandsperre nach Tabelle 8.8 erkennt man, dass der zweite Koeffizient Null beträgt. Somit kann der Multiplizierer für diese Konstante und der nachfolgende Addierer entfallen. Man erhält die Blockschaltbilder nach Bild 8.33. Die selben Überlegungen wie in den vorherigen Abschnitten bei den drei diskret einfach darstellbaren Frequenzen liefern die Ergebnisse nach Tabelle 8.11. Der normierte Amplitudengang und dessen doppeltlogarithmische Darstellung sind für beide Filtertypen in Bild 8.34 und Bild 8.35 aufgetragen.

Frequenz	0	$\frac{f_a}{4}$	$\frac{f_a}{2}$
$x[n]$	$\equiv 1$	$1; 0; -1; 0; 1; ...$	$1; -1; 1; -1; ...$
$y_{BP}[n]$	$\equiv 0$	$0; 1; 0; -1; 0; ...$	$\equiv$
$y_{BS}[n]$	$\equiv 1$	$\equiv 0$	$1; -1; 1; -1; ...$

Tabelle 8.11: Überlegungen zum Bandpass (BP) und zur Bandsperre (BS) zweiter Ordnung.

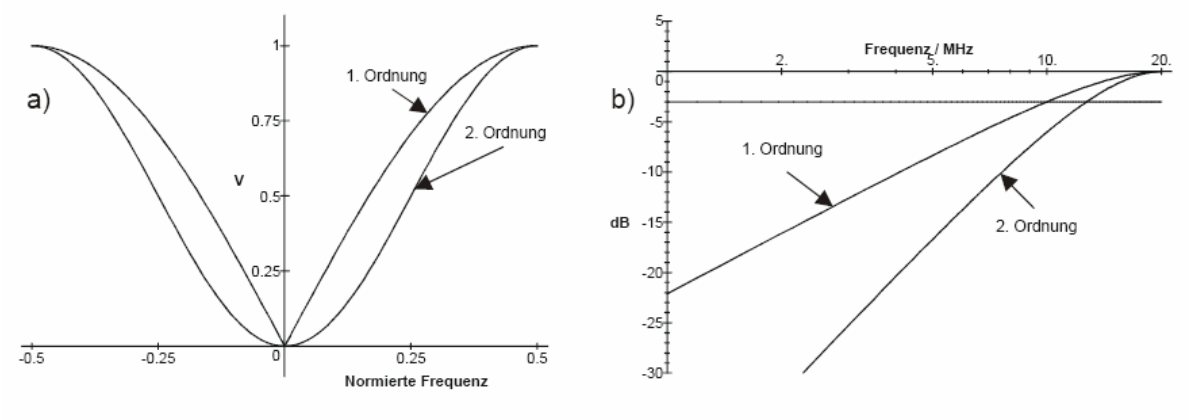

Bild 8.32: Amplitudengang des FIR–Hochpässe erster und zweiter Ordnung über der normierten Frequenz (a) und doppeltlogarithmischer Abtastfrequenz f_a = 40 MHz (b).

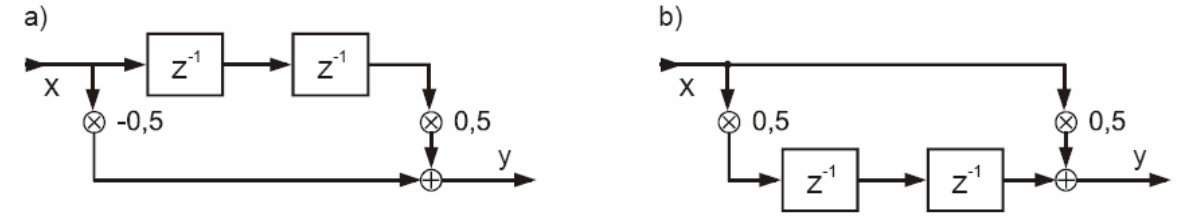

Bild 8.33: FIR–Bandpass zweiter Ordnung mit „globalem Summierer“ (a) und FIR–Bandsperre zweiter Ordnung mit „verteilten Summierern“ (b).

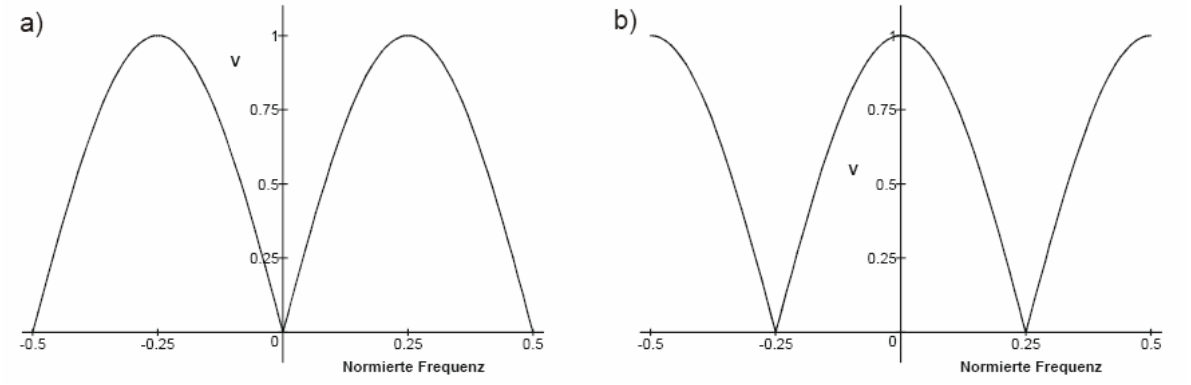

Bild 8.34: Normierter Amplitudengang für einen Bandpass zweiter Ordnung (a) und einer Bandsperre zweiter Ordnung (b).

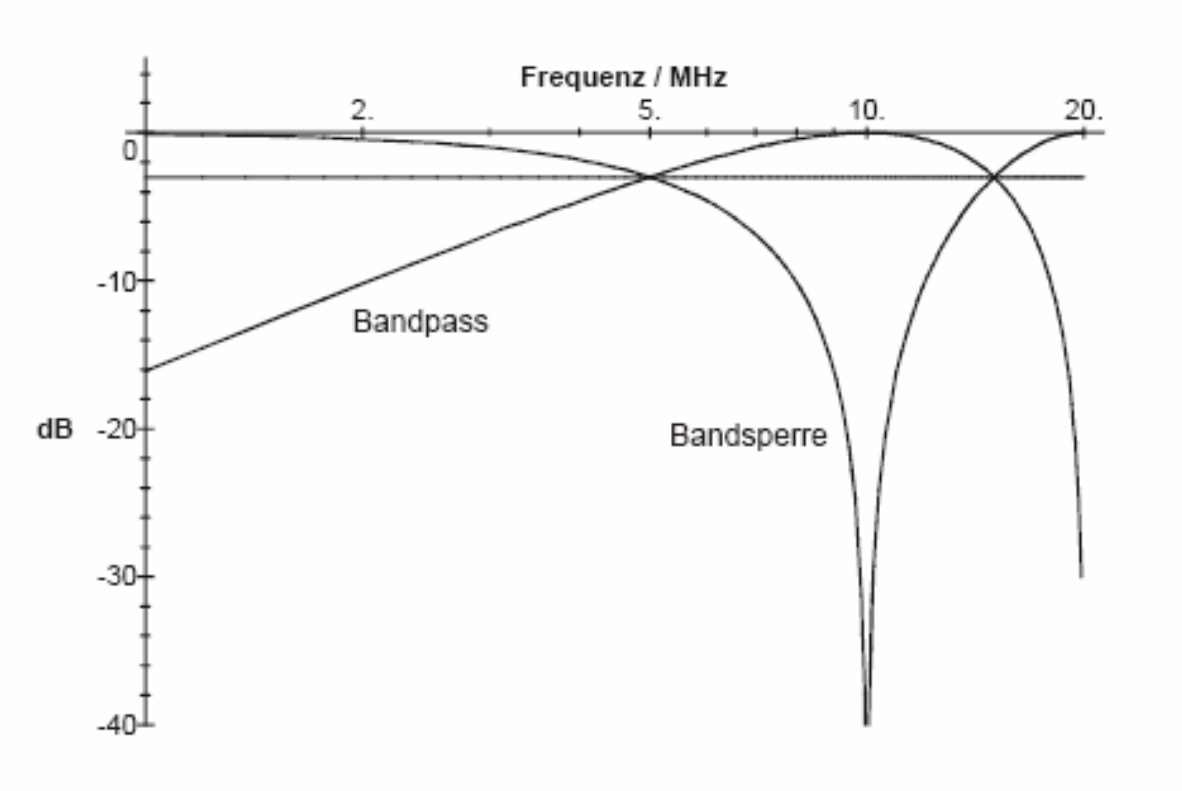

Bild 8.35: Doppeltlogarithmischer Amplitudengang von Bandsperre und Bandpass.

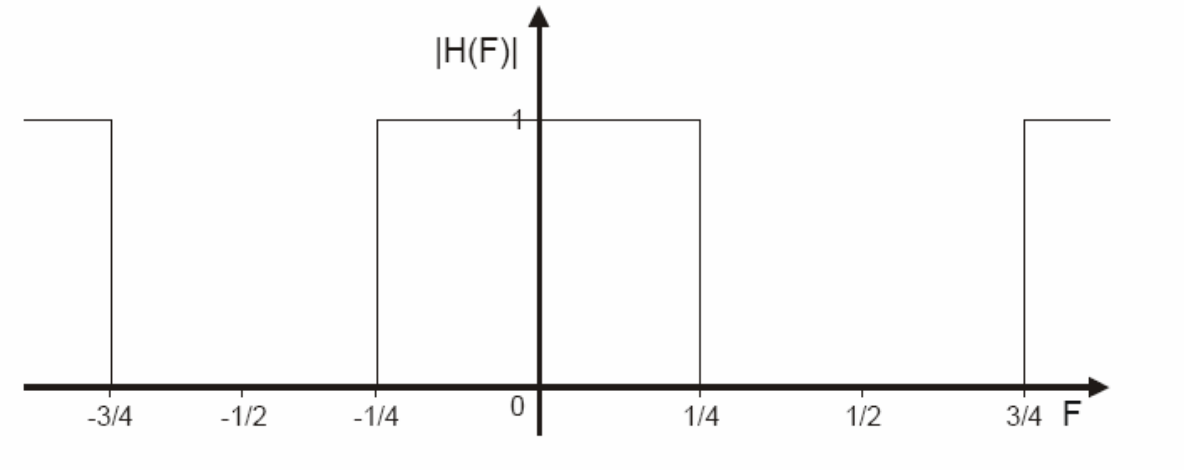

Bild 8.36: Amplitudengang eines idealen Tiefpasses.

8.1.3.12 Design von FIR–Filtern höherer Ordnung

Im Gegensatz zu den im vorherigen Kapitel behandelten einfachen Beispielen, bei denen die Koeffizienten vorgegeben waren und wir die Filtereigenschaften bestimmt haben, geht man normalerweise beim Design praxisrelevanter Filter den umgekehrten Weg. Dieser gliedert sich in drei Hauptabschnitte:

- Der Spezifikation der Filtereigenschaften.
- Der Approximation dieser Spezifikation mit einem LTD–System (Bestimmung der Filterkonstanten und eventuell der Filterordnung).
- Der Realisierung des Systems.

Das Ergebnis nach den ersten beiden Designschritten sind die Filterkoeffizienten des zu realisierenden Filters. Im Praktikum wird nur der letzte Designschritt durchgeführt. Im zweiten Versuch wird aber ein parametrisierter FIR–Filter entworfen, der eine Vielzahl von realen Filtern durch einfache Variation der Parameter ermöglicht. Zum Testen des Entwurfes werden die Filterkoeffizienten aus Tabellen entnommen. Im folgenden werden die ersten beiden Designschritte erläutert.

8.1.3.12.1 Spezifikation der Filtereigenschaften

Spezifiziert wird zum Einen der gewünschte komplexe Frequenzgang des Filters, wodurch Amplitudengang und Phasengang bestimmt sind. Darüber hinaus wird festgelegt, für welche Abtastfrequenzen der Filter gedacht ist, was allerdings erst bei der Realisierung eine Rolle spielt, nicht aber bei der Koeffizientenbestimmung. Als Beispiel werden wir einen FIR–Tiefpass mit linearer Phase spezifizieren. In Bild 8.36 ist der ideale Amplitudenverlauf aufgezeichnet ($f_g = 1/4\ f_a$). Dieser ist mit einem realen System nicht umsetzbar. Besonders die Unstetigkeitsstelle bei $F = \pm 1/4$ bereitet Probleme, da sie zu einer unendlich langen Impulsantwort führt. Aber auch der gerade Verlauf davor und

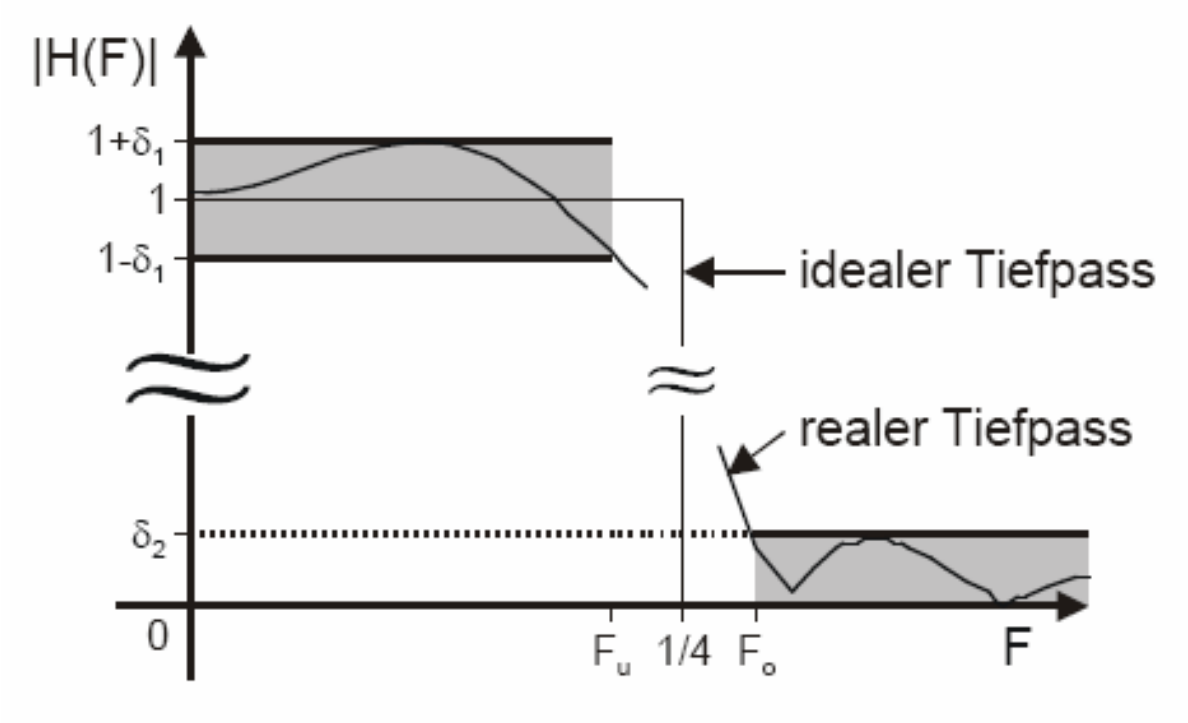

Bild 8.37: Toleranzschema Tiefpass.

danach sind nicht exakt umzusetzen. Deshalb muss ein Toleranzschema vorgegeben werden, innerhalb welchem sich der reale Filter zu befinden hat. In Bild 8.37 ist ein solches Schema für einen Tiefpass wiedergegeben. Zusätzlich ist der Amplitudengang des idealen Tiefpasses und eines realen Tiefpasses, welcher das Toleranzschema gerade erfüllt, eingezeichnet. Bezeichnend für ein solches Schema ist die zugelassene Abweichung δ_1 im Passband ($F = 0..F_u$), die Breite und Position des Übergangsbereichs ($F = F_u..F_o$) und die zugelassene Restverstärkung im Stoppband ($F > F_o$). Zusätzlich kann noch die genaue 3 dB Grenzfrequenz oder eine bestimmte Gleichspannungsverstärkung gefordert werden.

8.1.3.12.2 Approximation des Frequenzganges

Behandeln wird hier nur die anschauliche Fenstermethode. Weitere Methoden wie der Remez–Exchange Algorithmus sind in der Literatur zu finden. Es soll der ideale Tiefpass approximiert werden. Seinen Amplitudengang ist in Bild 8.36 zu sehen. Aus Kap. 8.1.3.5.1 kennen wir die Phasenverschiebung eines FIR–Filters (Formel 8.31). Der komplexe Frequenzgang ergibt sich somit zu:

$$H(f) = \begin{cases} e^{-j\pi f N T_a} & \text{für } -f_g \leq f \leq f_g \\ 0 & \text{sonst} \end{cases} \tag{8.34}$$

Daraus lässt sich mittels inverser Fouriertransformation die ideale diskrete Impulsant-

wort $h(n)$ bestimmen:

$$\begin{aligned} h(n) &= \int_{-f_a/2}^{f_a/2} H(f) \cdot e^{j2\pi f n T_a} \partial \\ f &= \int_{-f_g}^{f_g} e^{j\pi f T_a \cdot (2n-O)} \\ &= 2 f_g T_a \cdot \frac{\sin[\pi f_g T_a (2n-N)]}{\pi f_g T_a (2n-N)} \end{aligned} \tag{8.35}$$

Für die Werte $f_g = 1/8 \cdot f_a$ und $N = 31$ ist diese Impulsantwort in Bild 8.38 wiedergegeben. Sie ist unendlich lang und insbesondere akausal und kann somit nicht in einem realen System umgesetzt werden. Für einen realen Filter muss diese Folge abgebrochen werden. Damit der angenommene Phasenverlauf eintritt, muss der Filter von der Ordnung N sein und um den Wert $N/2$ symmetrisch sein. Die realisierbare Impulsantwort ergibt sich somit zu:

$$h_r^{(n)} = \begin{cases} h(n) & \text{für } 0 \leq n \leq N \\ 0 & \text{sonst} \end{cases} \tag{8.36}$$

Dies entspricht einer Multiplikation der idealen Impulsantwort mit einem Rechteckfenster von $n = 0$ bis $n = N$. Die Multiplikation im Zeitbereich entspricht der Faltung im Frequenzbereich und somit wird der ideale Amplitudengang verfälscht. Es entstehen Schwingungen im Pass- und Sperrband. Interessanterweise lassen sich diese Schwingungen durch eine erhöhte Filterordnung nicht reduzieren, lediglich der Übergangsbereich wird schmaler (siehe Bild 8.39). Dies wird als Gibbsches Phänomen bezeichnet. Eine Verringerung dieser Schwingungen kann man durch Anwendung einer anderen Fensterfunktion mit flacher abfallenden Rändern erreichen. Dies bewirkt allerdings immer eine Verbreiterung des Übergangsbereichs, so dass man für dieselbe Steilheit eine höhere Filterordnung benötigt. Gängige Fensterfunktionen sind das Hamming, Hanning, Blackman und Kaiser Fenster. Ist die genaue Grenzfrequenz für den Filter von Bedeutung, so muss in mehreren Iterationsschritten f_g in Formel 8.35 angepasst werden, bis der resultierende Amplitudengang seinen Wert von -3 dB an der Grenzfrequenz erreicht. Anschließend werden die Koeffizienten normiert. Wird eine Gleichspannungsverstärkung von 1 gefordert, so muss die Summe aller Koeffizienten 1 ergeben (siehe Kap. 8.1.3.5.1), die endgültigen Koeffizienten ergeben sich somit aus:

$$\alpha_i = \frac{\alpha_{i,f}}{\sum_{l=0}^{N} \alpha_{l,f}} \quad i = 0...N \tag{8.37}$$

Hierin sind die Koeffizienten vor der Normierung mit $\alpha_{i,f}$ bezeichnet. Probleme können sich ergeben, wenn am Eingang ein harmonisches Signal mit der Frequenz des Maximums des Filteramplitudenganges anliegt. In Bild 8.39 ist zu erkennen, dass die Verstärkung dort größer 1 ist. Dies kann zu einem Modulo 2 Überlauf am Ausgang des FIR–Filters führen (siehe Kap. 8.1.2.4.4) und somit zu völlig falschen Ergebnissen. Um diesen Fall ausschließen zu können, bietet sich die Normierung der Konstantenbeträge auf 1 an. Allerdings muss in diesem Fall nach dem Filter mit einem Verstärker (Multiplizierer) die Gleichspannungsverstärkung korrigiert werden. Als Beispiel zeigt Bild 8.40

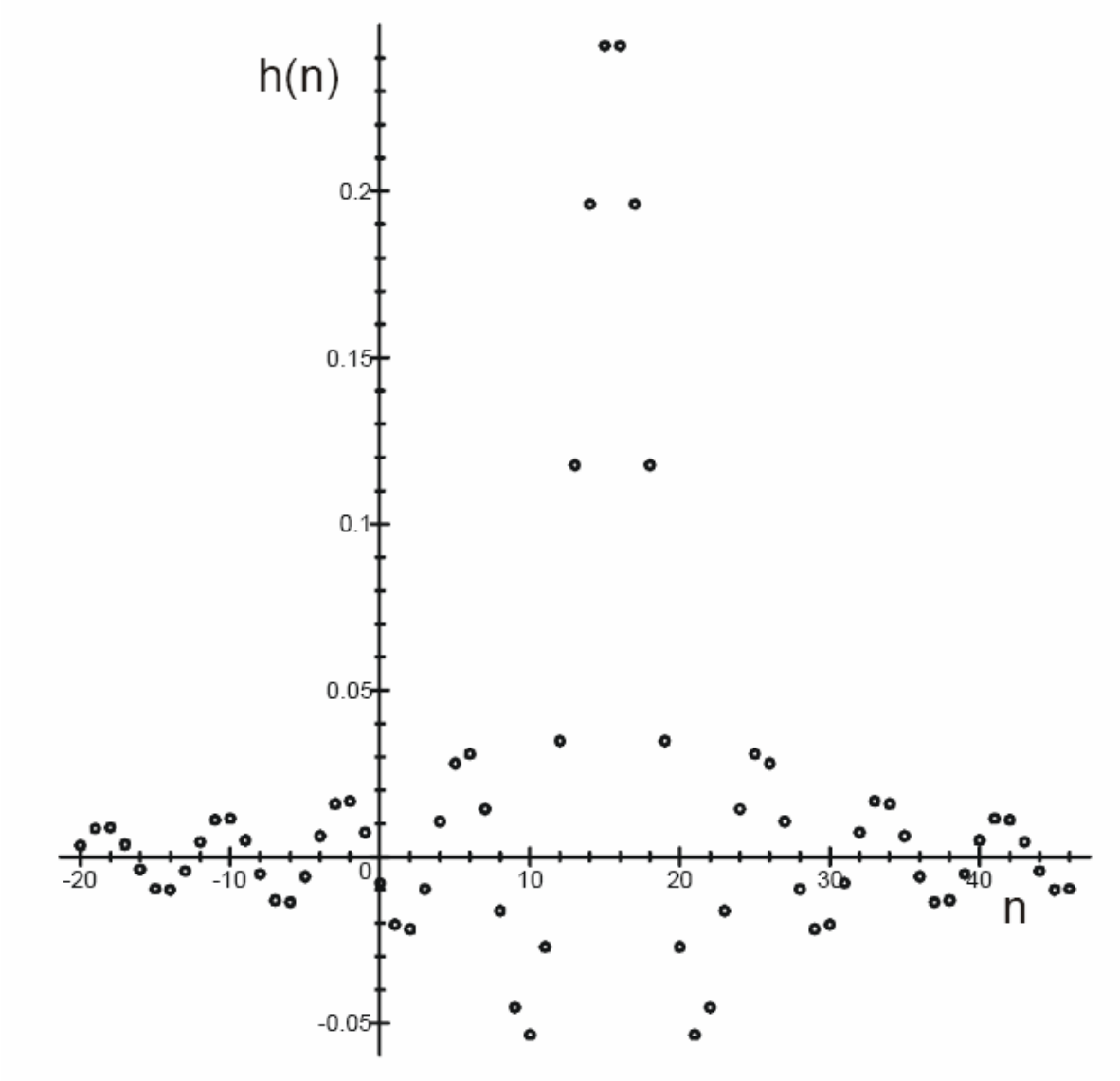

Bild 8.38: Ideale Impulsantwort für $f_g = 1/8\ f_a$ und $N = 31$.

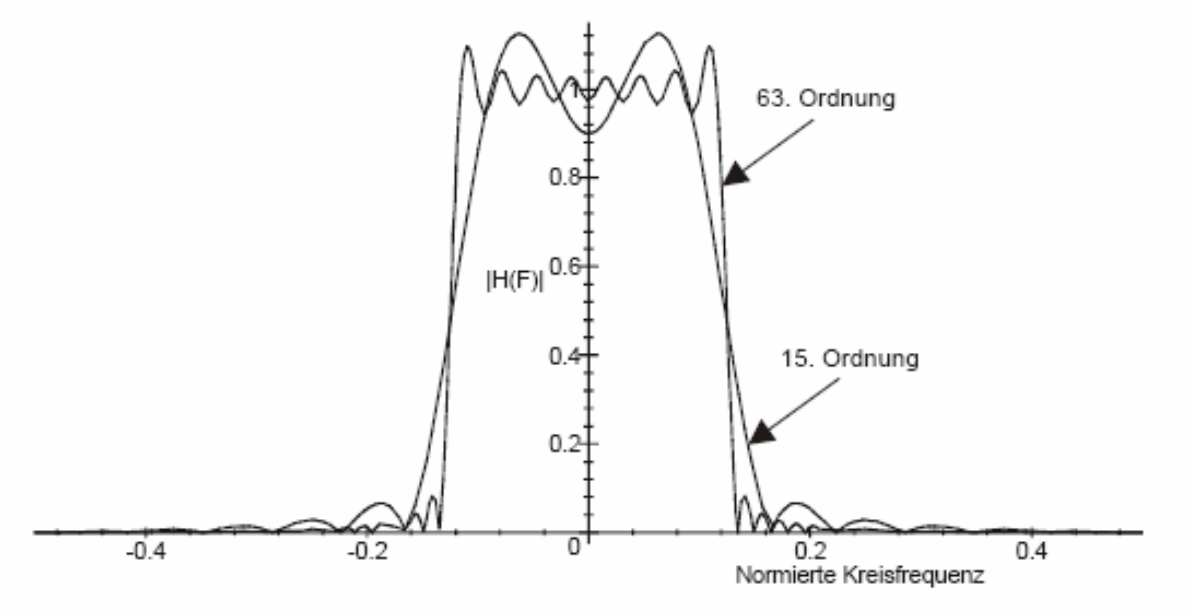

Bild 8.39: Amplitudengang rechteckgefensterter FIR–Filtern.

zwei FIR–Filter 15. Ordnung, der eine wurde mittels Rechteckfenster, der andere mit dem Hamming Fenster ermittelt. Beide wurden auf eine Gleichspannungsverstärkung von 1 normiert.

8.2 Versuche mit FIR–Filtern zweiter Ordnung

Verwenden Sie für die Erstellung Ihrer Projekte das Device EP2C70F672C6 wie im Versuch LED–Lauflicht. Setzen Sie auch hier alle nicht verwendeten Pins auf tri–state! Dokumentieren Sie Ihre Schaltungen für den abzugebenden Praktikumsbericht möglichst

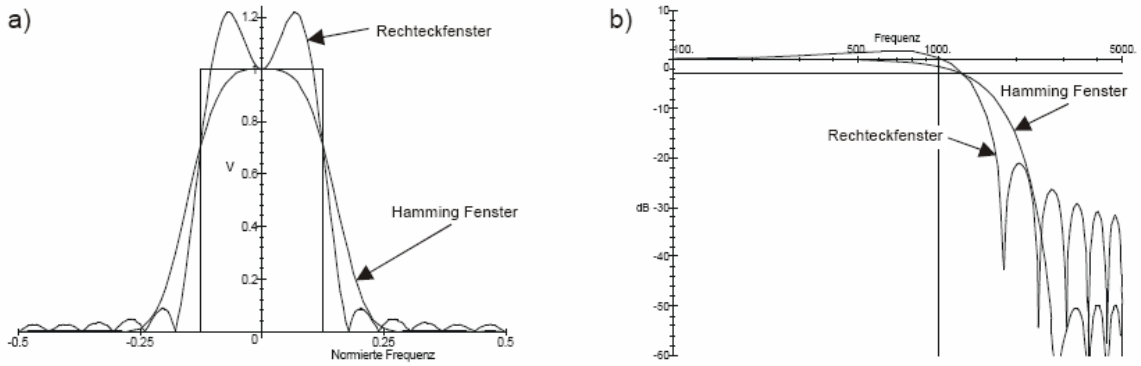

Bild 8.40: Vergleich Rechteck/Hamming Fensterung ($f_g = 1/8\ f_a$) über der normierten Frequenz (a) und doppeltlogarithmisch (b) aufgetragen.

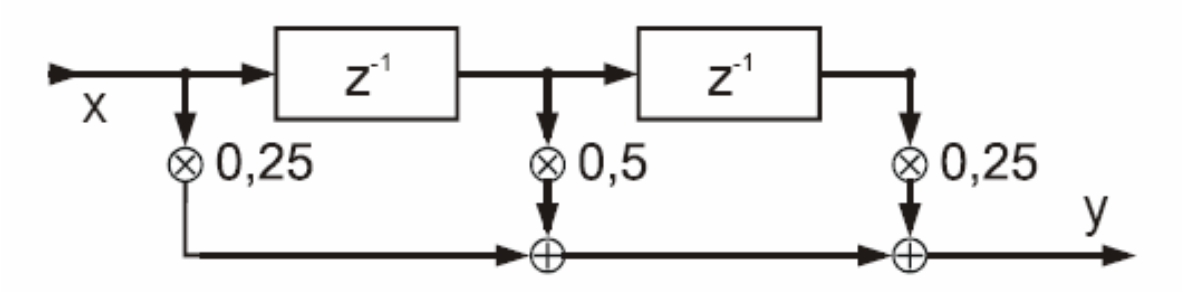

Bild 8.41: FIR–Tiefpass 2. Ordnung mit globalem Summierer.

ausführlich (Screenshots, Ergebnisse!) auch für die einzelnen Subschaltungen.

8.2.1 Tiefpass erstellen

Erstellen Sie einen FIR–Tiefpass zweiter Ordnung mit **globalem Summierer** mittels grafischer Schaltungseingabe. In Bild 8.41 ist das Blockschaltbild angegeben. Der Eingang vom A/D–Wandler (x) ist wie bereits erwähnt im Zweierkomplement codiert. Der D/A–Wandler am Ausgang erwartet seine Werte ebenfalls in diesem Format. Für die Multiplikation und die Addition können Sie Megafunctions verwenden, achten Sie aber auf deren Funktionalität.

- Kompilieren Sie den Filter und bestimmen Sie den Logikzellenverbrauch und die maximale Arbeitsfrequenz.
- Simulieren Sie Ihren Filter, indem sie seine Impuls- und Sprungantwort aufzeichnen und mit dem erwarteten Ergebnis vergleichen. Informationen über die diskrete Impulsantwort finden Sie in Kap. 8.1.2.2.1.
- Dokumentieren Sie Ihre Ergebnisse für den Praktikumsbericht.
- Abschließend erstellen sie ein Symbol für die leichte Wiederverwendung des Filters als Modul in einer übergeordneten Schaltungsebene.

Hinweis: Kopieren Sie nicht Ihren Tiefpass, um nur die Anordnung der Elemente und die Koeffizienten zu ändern, da Sie sonst Probleme mit den Namen der Elemente beim Zusammenfügen in die übernächste Aufgabe bekommen!

8.2.2 Hochpass erstellen

Erstellen Sie im Schematics–Editor das Schaltbild eines FIR–Hochpasses 2. Ordnung, diesmal verwenden Sie allerdings die Elementanordnung mit **verteilten Summierern**. Die verschiedenen Elementanordnungen sind in Kap. 8.1.3.2.3 beschrieben. Die Koeffizienten für alle vier Filtertypen sind in Tabelle 8.8 zu finden. Simulieren Sie Ihren Hochpass wieder mittels Impuls- und Sprungantwort.

- Kompilieren Sie den Filter und bestimmen Sie den Logikzellenverbrauch und die maximale Arbeitsfrequenz.
- Vergleichen Sie den Ressourcenverbrauch und die maximale Arbeitsfrequenz Ihres Tiefpasses und Hochpasses.
- Welche Variante scheint vorteilhafter für den Gebrauch mit programmierbarer Logik zu sein?
- Dokumentieren Sie Ihre Ergebnisse für den Praktikumsbericht.
- Erstellen Sie abschließend ein Symbol für die leichte Wiederverwendung des Filters als Modul in einer übergeordneten Schaltungsebene.

Hinweis: Kopieren Sie nicht Ihren Tiefpass, um nur die Anordnung der Elemente und die Koeffizienten zu ändern, da Sie sonst Probleme mit den Namen der Elemente beim Zusammenfügen in die übernächste Aufgabe bekommen!

8.2.3 Bandpass

Realisieren Sie einen FIR–Bandpass 2. Ordnung mit VHDL zur Schaltungsbeschreibung. Verwenden Sie einen **globalem Summierer**. Die entsprechenden Filterkoeffizienten finden Sie in Tabelle 8.8.

- Kompilieren Sie den Filter und bestimmen Sie den Logikzellenverbrauch und die maximale Arbeitsfrequenz.
- Vergleichen Sie den Ressourcenverbrauch und die maximale Arbeitsfrequenz Ihres Bandpasses.
- Dokumentieren Sie Ihre Ergebnisse für den Praktikumsbericht.
- Erstellen Sie ein Symbol für die leichte Wiederverwendung des Filters als Modul in einer übergeordneten Schaltungsebene.

8.2.4 Bandsperre

Realisieren Sie eine FIR–Bandsperre 2. Ordnung mit VHDL zur Schaltungsbeschreibung. Verwenden Sie einen **verteiltem Summierer**. Die entsprechenden Filterkoeffizienten finden Sie in Tabelle 8.8.

- Kompilieren Sie den Filter und bestimmen Sie den Logikzellenverbrauch und die maximale Arbeitsfrequenz.
- Vergleichen Sie den Ressourcenverbrauch und die maximale Arbeitsfrequenz Ihrer Bandsperre.
- Welche Variante (Bandpass & Bandsperre) scheint vorteilhafter für den Gebrauch mit programmierbarer Logik zu sein?

- Dokumentieren Sie Ihre Ergebnisse für den Praktikumsbericht.
- Erstellen Sie abschließend ein Symbol für die leichte Wiederverwendung des Filters als Modul in einer übergeordneten Schaltungsebene.

8.2.5 Projekt mit allen Filtern

Erstellen Sie ein Projekt, in welchem sie alle vier Filter instantiieren. Ob Sie das Projekt in VHDL oder als Schaltplan umsetzen ist irrelevant. Da es aber hauptsächlich um die Verdrahtung der einzelnen Filter geht, bietet sich die Schaltplan–Variante an. Alle Filter werden von einem Eingang (A/D–Wandler) gespeist. Der Ausgang (D/A–Wandler) soll mittels einem Taster zwischen den vier möglichen Filtern umschaltbar sein. Der momentane Betrieb (Filtertyp) soll sichtbar sein. Verwenden Sie dazu entweder zwei LED's im Binärcode oder die Sieben–Segment–Anzeige.

- Kompilieren Sie das Projekt und bestimmen Sie den Logikzellenverbrauch.
- Simulieren Sie das Projekt, so dass sie von jedem Filter einmal die Impulsantwort erhalten.
- Dokumentieren Sie Ihre Ergebnisse für den Praktikumsbericht.

Hinweis: Beachten Sie die Regeln zum Einbinden von untergeordneten Projekten (Einzelfilter) in ein größeres Projekt.

8.2.6 Programmierung des Cyclone II Bausteins

Vor dem Programmieren der Schaltung müssen noch die Ein- und Ausgänge auf die entsprechenden Pins des Cyclone II Bausteines gesetzt werden. Der Eingang Ihrer Filter sollte auf den A/D–Wandler A und der Ausgang soll auf den D/A–Wandlerausgang A gelegt werden. Kontrollieren Sie den Jumper J19 auf OSC, was den Systemtakt an die Wandlerbausteine legt.

- Kompilieren Sie das Projekt und bestimmen Sie den Logikzellenverbrauch.
- Simulieren Sie das Projekt, so dass sie von jedem Filter einmal die Impulsantwort erhalten.
- Programmieren Sie den Baustein.
- Schließen Sie den Funktionsgenerator an den gewählten Eingang des A/D–Wandlers an.
- Verwenden Sie den SignalTapAnalyzer in Quartus als Oszilloskop zur Überprüfung der korrekten Filterfunktionen.
- Dokumentieren Sie Ihre Ergebnisse für den Praktikumsbericht.