FH D Digitaltechnik 2

Prof. Dr. J.Wietzke

Email	j.wietzke@fbi.fh-darmstadt.de
Tel	+49 (6151) 16-8472
Fax	+49 (6151) 16-8935
Post	Haardtring 100, 64295 Darmstadt

Sprechstunde Do. 12.00-13.00 Uhr, 14/208

9 Einleitung

9.1 Vorwort

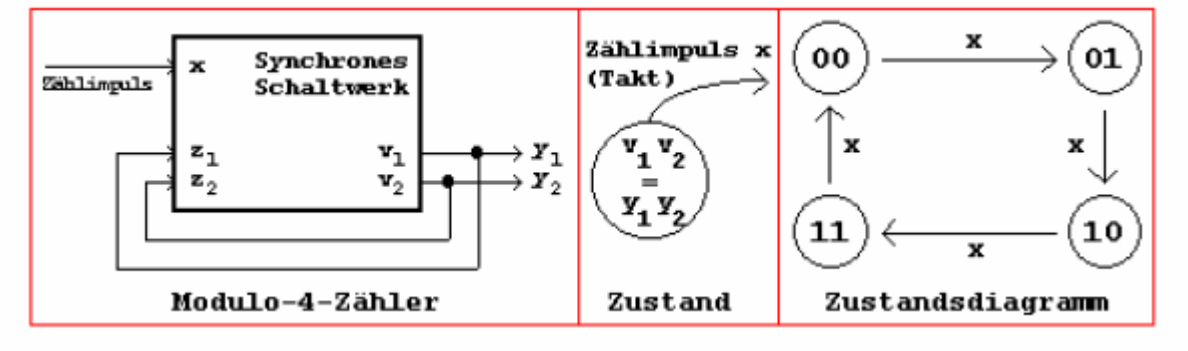
Das Skript wurde aus mehreren Vorlagen und eigenen Texten zusammengestellt, auch zur möglichen Vereinheitlichung der Vorlesungen, die über mehrere Züge von verschiedenen Dozenten vorgetragen werden.
Das Skript ist nicht zur Weitergabe bestimmt sondern dient nur zur eigenen Vorbereitung der Vortragenden. Teil 1 endet vor oder mit der Einführung der Flip-Flops, je nach Fortschritt der Vorlesung, dann beginnt Teil 2 mit Fokus auf Schaltwerken, Speichern und Zustandsautomaten. Da die Vorlesung Rechnerorganisation thematisch recht voll ist, wird die Datensicherung mit Mitteln der Kodierung hier mit der Bearbeitung der Massenspeicher besprochen.

10 Sequentielle Grundschaltungen, Schaltwerke

Die bisherigen kombinatorischen Schaltungen wandeln eine Kombination von Eingangswerten direkt in eine Kombination der Ausgangswerte. Die Ausgangswerte kombinatorischer Schaltungen hängen nur von den aktuellen Eingangswerten, nicht aber von vergangenen Ein- oder Ausgangswerten ab. Solche Art Schaltungen bezeichnet man auch als Schaltungen ohne Gedächtnis.
Möchte man Werte aus der Vergangenheit in einer digitalen Schaltung speichern, benötigt man jedoch eine Schaltung mit Gedächtnis. Diese Schaltungen bezeichnet man als sequentielle Schaltungen. Sequentielle Schaltungen erhält man dadurch, daß man Ausgangssignale einer Schaltung auf ihren Eingang zurückkoppelt. Bei direkter Rückkopplung von Ausgangssignalen einer kombinatorischen Schaltung auf ihre Eingänge spricht man von einer asynchron rückgekoppelten Schaltung oder auch einer asynchron sequentiellen Schaltung. Sie besitzt somit die folgende Struktur:

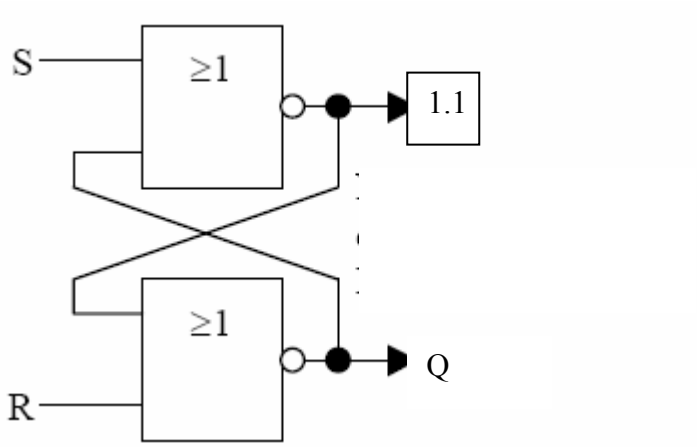
10.1 Sequentielle Schaltung (Schaltwerk):

• Ausgangswert = f (Eingangswerte, Zeit)
• Enthält Speicherelemente („Gedächtnis“)
• Funktionsbeschreibung durch Zustandstabelle, Zustandsgraf

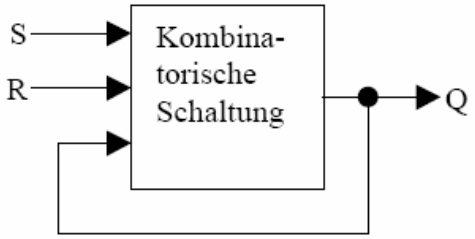


10.1.1.1 RS-Latch

Betrachten wir als erstes die folgende Schaltung aus zwei Invertern und zwei ODER-Gattern mit Rückführungen.
Bei Eingabe R = 1 und S = 0 erzeugt die Schaltung die Ausgabe Q = 0. Ist R = 1, dann macht der obere Inverter daraus eine 0, die wiederum vom unteren Inverter in eine 1 zurückverwandelt wird. Diese 1 bewirkt wieder Q = 0. Wechselt nun der Eingabewert R nach 0, bleibt Q = 0. Der Wechsel macht sich also nicht am Ausgang bemerkbar. In anderen Worte, die Schaltung speichert das 0-Bit. Man sagt auch die Schaltung hat den Zustand 0 angenommen. Und umgekehrt, bei Eingabe R = 0 und S = 1 speichert die Schaltung das 1-Bit, geht in den Zustand 1. Das heißt, wir können mit dieser Schaltung ein Bit speichern - und für n Bits benötigen wir n Stück davon.
Eingabe R=1 und S=0 wird als Reset, Eingabe R=0 und S=1 wird als Set bezeichnet.
Setzen wir dagegen R = S = 1, nehmen sowohl Q als auch Q den Wert 0 an, und es ist nicht vorhersagbar, welcher Zustand angenommen wird, wenn danach R = S = 0 gesetzt wird. Bei realen Gattern hängt das von den stets vorhandenen Asymmetrien ab. Die Eingabe R = S = 1 (das bedeutet ja gleichzeitig Rücksetzen und Setzen) ist also zu vermeiden, wenn diese Schaltung als Speicherelement verwendet werden soll. Man nennt es RS-Latch.

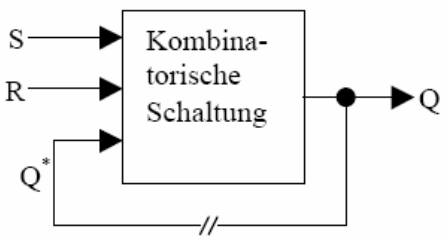


NOR-RS-FlipFlop
Die Schaltung gehört zur Familie der bistabilen Kippstufen.
Will man diese Schaltung mit einer Wahrheitstabelle beschreiben, so muß man den Ausgangswert Q auch auf der Eingangsseite mit aufführen, denn beim Halten von Q hängt der Ausgangswert von Q sowohl von S und R als auch vom aktuellen Wert Q ab. Die Schaltung lässt sich somit durch die folgende Struktur beschreiben:



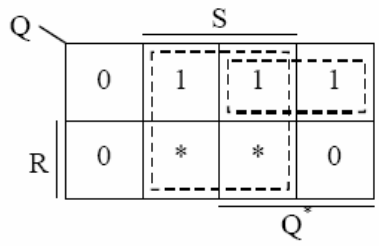
Sie besteht also aus einer kombinatorischen Schaltung, die frei von Rückkopplungen ist, und einer externen Rückkopplung des Ausgangs Q auf den Eingang.

Zum Entwurf der Schaltung kann man gedanklich die Rückkopplung auftrennen und den Ausgang Q auf der Eingangsseite mit Q* bezeichnen:

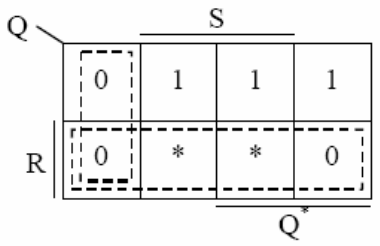


S	R	Q*	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	*
1	1	1	*

Disjunktive Form



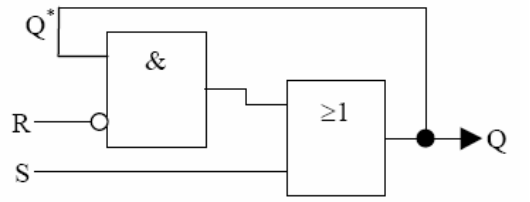
Konjunktive Form

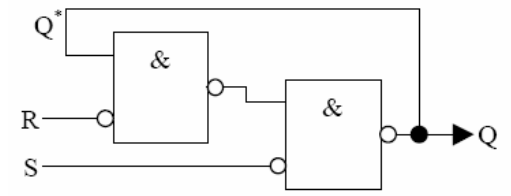


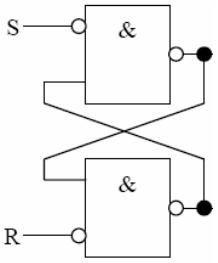
$Q = S \veebar ((\neg R) \wedge Q^*)$ Disjunktive Minimalform
$Q = (\neg R) \barwedge (S \veebar Q^*)$ Konjunktive Minimalform

Das Umsetzen der Gleichungen in Schaltungen liefert:

Schaltung zur disjunktiven Minimalform

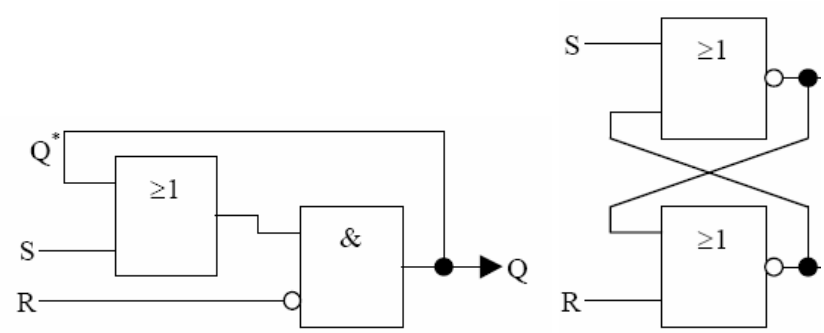




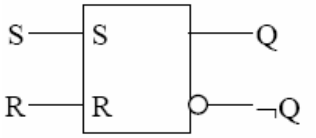


NAND-RS-Flip-Flop

Schaltung zur konjunktiven Minimalform



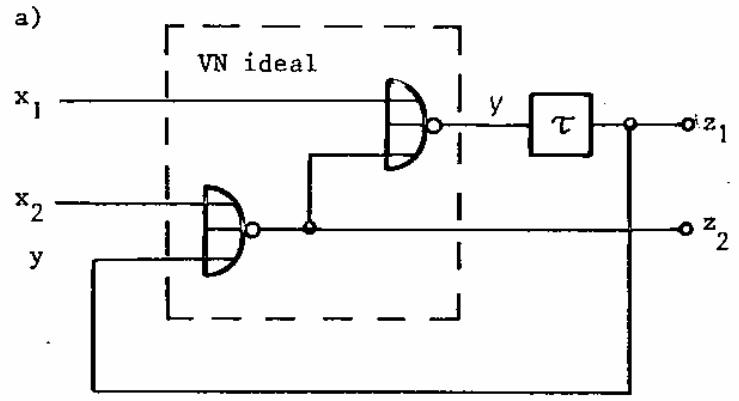
Für alle Varianten des RS-Flip-Flops gilt das nachfolgende Schaltsymbol:



Bei Spezifikation des RS-Flip-Flops mit diesem Schaltsymbol ist das Verhalten bei der Eingangskombination R,S=11 nicht spezifiziert. Setzt man ein RS-Flip-Flop ein, sollte entweder diese Eingangskombination vermieden werden, oder die verschiedenen Ausgangswerte unterschiedlicher Realisierungen sind akzeptabel.

Eine andere Betrachtungsweise statt des Auftrennens ist die Einführung einer Gatterverzögerungszeit:
Dem wie bisher idealen Gatter am Ausgang wird noch vor der Rückkopplung ein Verzögerungsglied eingefügt. Dann läßt sich ebenfalls ein Ausgangssignal aus der reinen Kombinatorik der Eingangssignale berechnen und erst nach einer kleinen Verzögerung bewirkt die Rückkopplung eine Veränderung des Signals Q* und damit ggf. auch eine weitere Veränderung des Ausgangs (kippen).
Welche Zustände stabil sind und welche durch die Rückkopplung kippen, läßt sich im KV- Diagramm oder in der Logiktabelle erkennen. Haben das Signal Q* und das sich ergebende Ausgangssignal Q den gleichen logischen Wert, so handelt es sich um einen stabilen Zustand. Sind sie unterschiedlich, so wird die Schaltung nach der genannten kurzen Verzögerung in den neuen Zustand kippen.
In untenstehendem Beispiel sind die stabilen Zustände eingekreist, es handelt sich also erwartungsgemäß um eine bistabile Schaltung.

a)



b) $Y = ((x_2 \vee y)' \vee x_1)' = x_1' x_2 \vee x_1' y$

$z_1 = y$

$z_2 = (x_2 \vee y)' = x_2' y'$

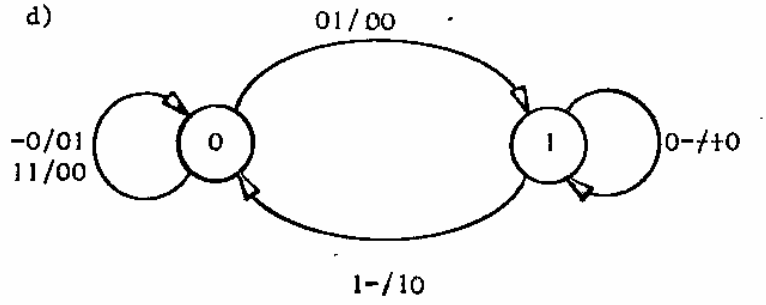
c)

	$x_1 x_2$				$x_1 x_2$			
y	00	01	11	10	00	01	11	10
0	(0)	1	(0)	(0)	01	00	00	01
1	(1)	(1)	0	0	10	10	10	10
				y				$z_1 z_2$

Übergangs-tabelle

Ergebnis-tabelle

d)

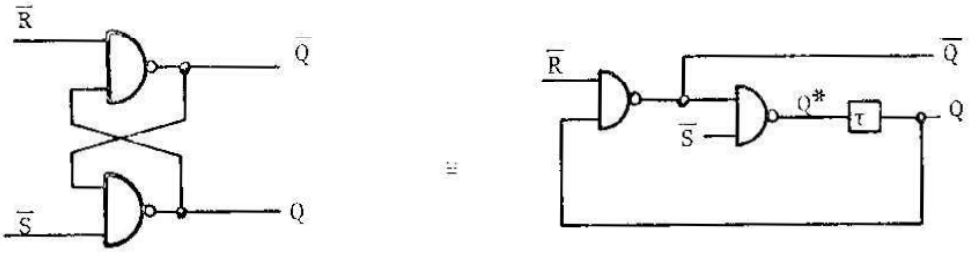


Notation:

$x_1 x_2 / z_1 z_2$

Die gleichen Betrachtungen für NAND-RS-FlipFlops

a)



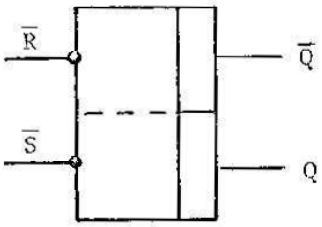
b) $Q^* = ((\overline{R}Q)'\overline{S})' = \overline{R}Q \vee \overline{S}'$

c)

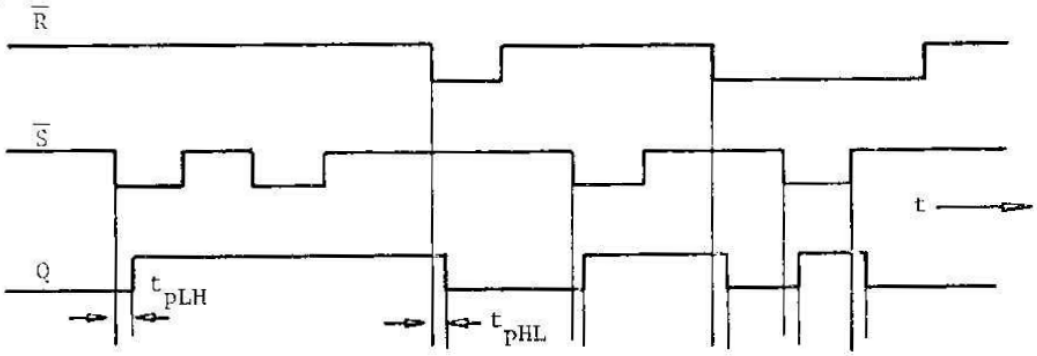
Q	R̄S̄ = 00	01	11	10
0	1	0	0	1
1	1	0	1	1

Q^*

d)



e)



RS-NAND-FF a) Schaltung, b) Zustandsgleichung, c) Zustandstabelle, d) Schaltungssymbol, e) typisches Verhalten

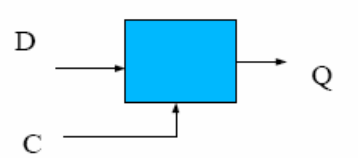
In einem späteren Kapitel werden dann Läufe und Hazards, die in solchen Schaltungen auftreten können, besprochen werden.

Die einfachsten Speicherelemente können nur binäre Werte 0 oder 1 aufnehmen, speichern und wieder abgeben - man sagt auch die Zustände 0 oder 1 einnehmen. Aus solchen Speicherelementen lassen sich aber komplexere Speicherelemente aufbauen, die ganze Bitfolgen speichern können. Es gibt mehrere Typen einfacher Speicherelemente. Man nennt sie allgemein Flip-Flops (FFs).

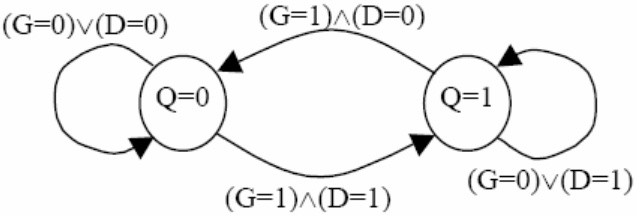
10.1.1.2 Pegelgesteuerte D-Flip-Flop

Die einfachsten Flip-Flops sind die sogenannte Latches (Riegel).

C ist der Steuer-, D der Datenein- und Q der Ausgang.

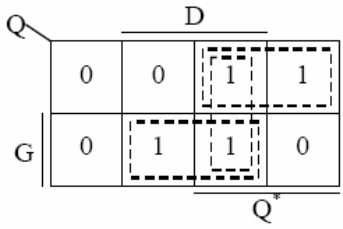


Bei diesem Latch ist im Load-Zustand (C = 1) der Eingangwert D am Ausgang Q sofort sichtbar. Man spricht deshalb auch von einem „transparenten Flip-Flop“. Der Ausgang folgt den Änderungen am Eingang, solange der Steuereingang C den Wert 1 hat. Bei Wechsel zu 0 wird der augenblickliche Eingangswert gespeichert.
Ein D-Flip-Flop besitzt einen Dateneingang. Der Wert dieses Eingangs wird in das D-Flip-Flop eingeschrieben, wenn die Freigabebedingung erfüllt ist. Beim pegelgesteuerten D-Flip-Flop ist die Freigabebedingung G=1.
Damit lässt sich das Zustandsdiagramm des Flip-Flops aufstellen:



Man spricht in diesem Zusammenhang auch von einem pegel- oder taktzustands-gesteuerten Speicherelement (Gemeint ist der Pegel/Zustand des Steuersignals C). (Bei Taktflanken-Steuerung wird die anliegende Information nur während der Taktflanke übernommen.) Das Latch (D-Latch) ändert also seinen Zustand nur, wenn das Steuersignal C (Taktsignal) den Wert 1 hat. Dann ist sein Zustand gleich D. Der Eingabewert wird gespeichert (verzögert, delayed). Der gespeicherte Wert (Zustand) steht am Ausgang permanent zur Verfügung.

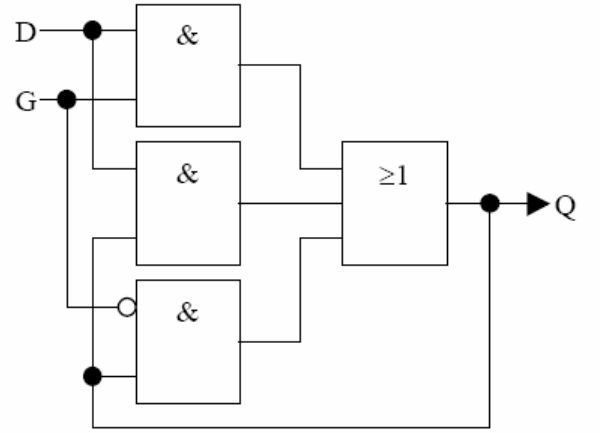
Q^*	G	D	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1



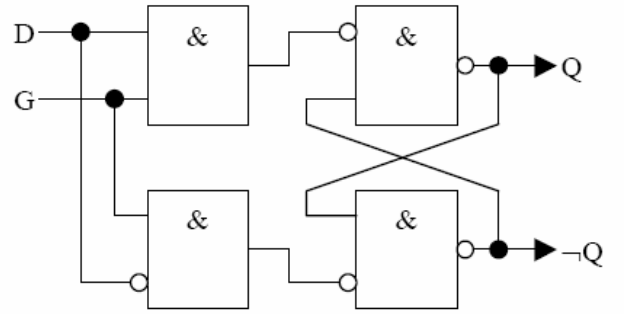
Die gezeigten Primimplikanten werden alle in die Gleichung aufgenommen, um Hazardfreiheit (folgt später) zu gewährleisten. Damit ergibt sich die Gleichung des Flip-Flops zu:

$Q = (D \wedge G) \vee (D \wedge Q^*) \vee ((\neg G) \wedge \vee Q^*)$

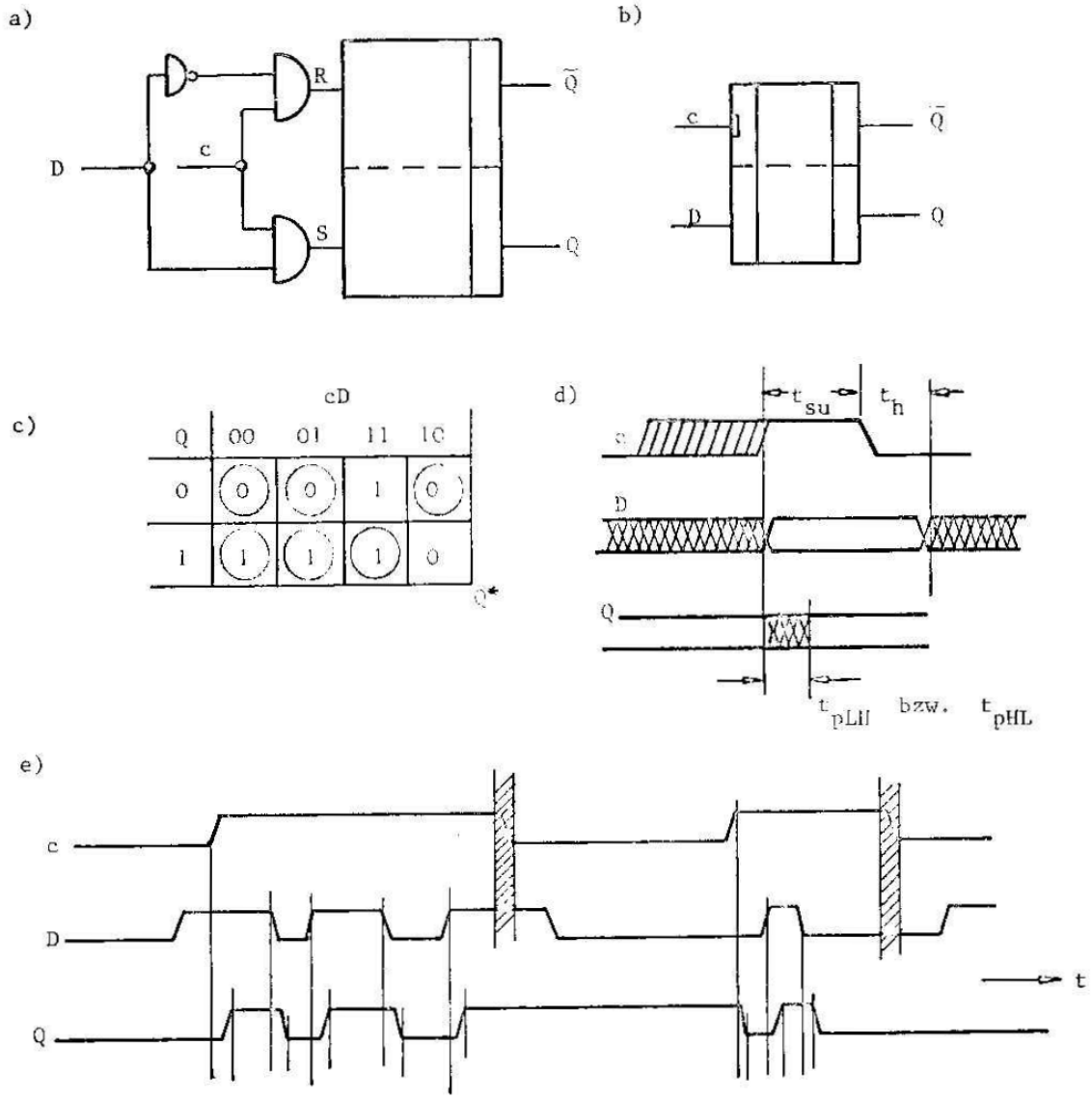
Die Realisierung der Gleichung führt zu folgender Schaltung:



Das pegelgesteuerte D-Flip-Flop kann auch dadurch hergeleitet werden, daß man beim pegelgesteuerten RSFlip-Flop den S-Eingang direkt und den R-Eingang invertiert mit dem D-Signal verbindet:



Vorteil dieser Schaltung gegenüber der ersten ist der etwas geringere Schaltungsaufwand (8 gegenüber 9).
Nachteil ist jedoch, daß maximal 3 Gatterstufen zwischen dem D-Eingang und dem Q-Ausgang liegen. Die erste Schaltung kommt hingegen mit zwei Stufen aus und ist somit schneller.

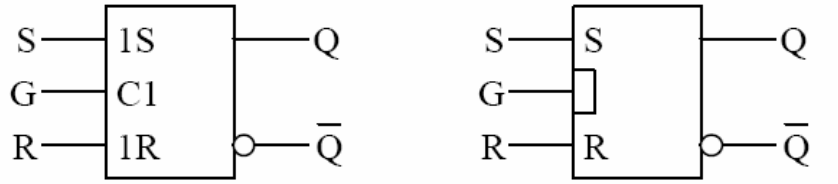


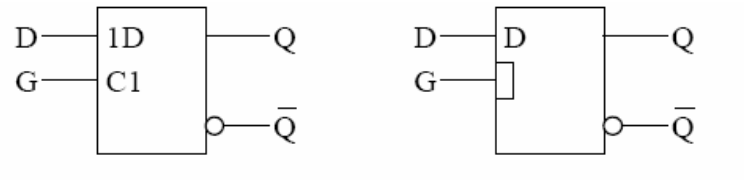
D-FF, a) Schaltung, b) Schaltungssymbol, c) Zustandstabelle, d) Zeitparameter, e) typischer Verlauf von Ein- und Ausgangs-Signalen

Bei der Anwendung müssen die Zeitparameter tsetup und thold beachtet werden. Eine gewisse Zeit vor der abfallenden Flanke des Taktpegelsignals c muß das Datensignal D stabil sein und nach dieser Flanke muß der Pegel an D für eine gewisse Hold-Zeit stabil bleiben, wenn dieser Pegel sicher übernommen werden soll. Die Tatsache, daß sich bei ungepufferten, taktpegelgesteuerten FFs das Ausgangssignal Q schon ändern kann, während am Eingang Taktpuls und Daten D bzw, R und S noch konstant anliegen müssen, hat Konsequenzen für die Gestaltung des Informationstransportes zwischen Speichergliedern in Schaltwerken.
Eine Antwort auf das Problem sind Vorspeicher-FFs, sogenannte Master-Slave-FFs, die den Datentransport zweiphasig vornehmen. Diese werden weiter unten besprochen.

Generell gilt:
Eine Mißachtung der Randbedingungen tsetup und tHold kann zu metastabilen Zuständen der Schaltwerke führen, die einige Millisekunden andauern können!

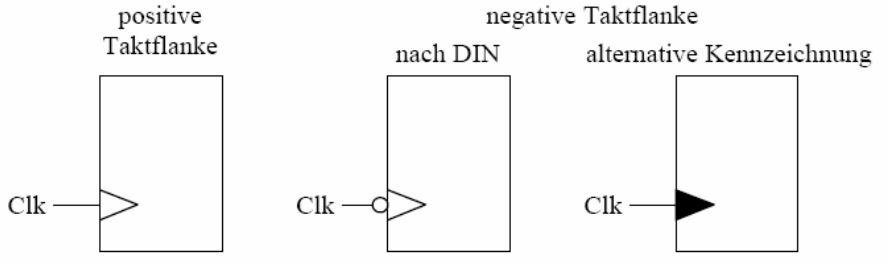
Nachfolgend ist links das DIN-Symbol dargestellt. Rechts ist ein gebräuchliches Blockschaltbildsymbol gezeigt.



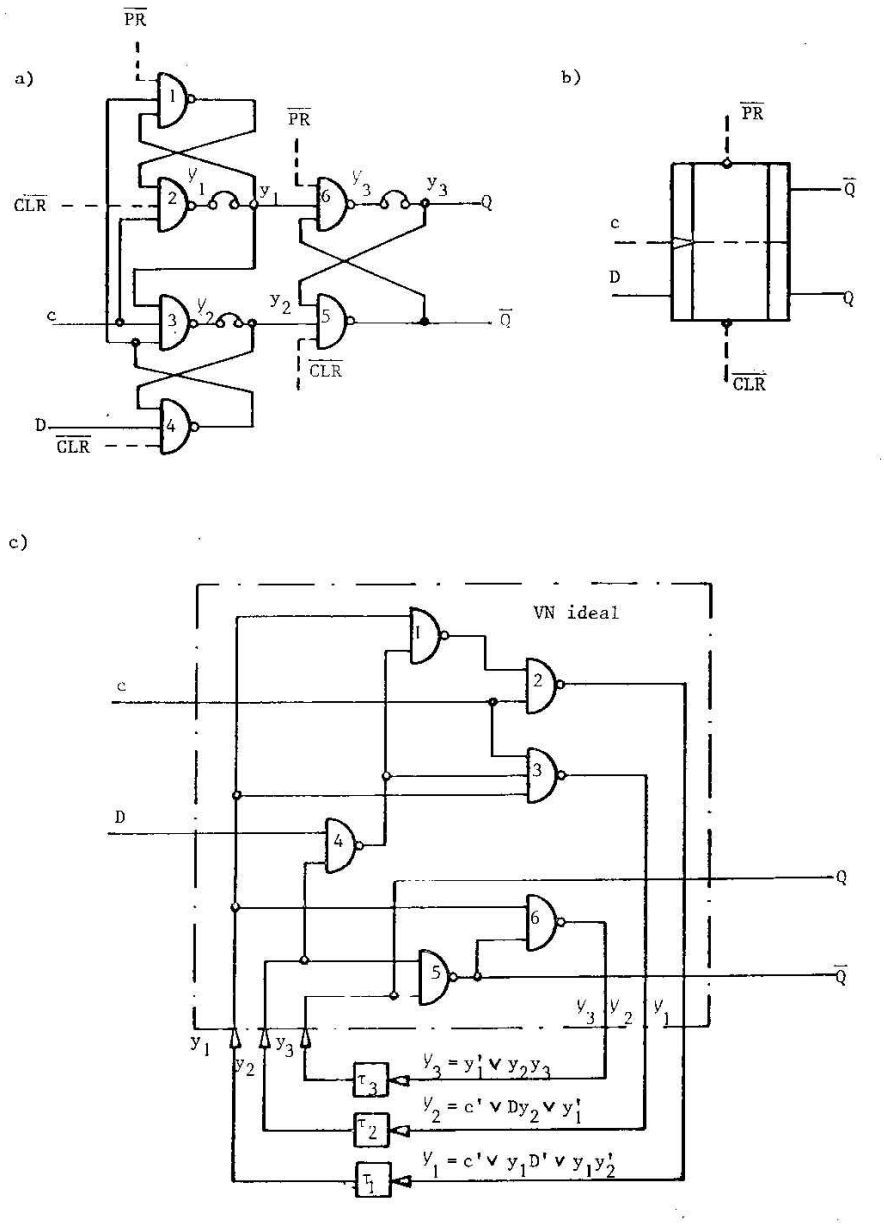


10.1.1.3 Flankengesteuerte Flip-Flops

Flankengesteuerte Flip-Flops arbeiten synchron zu einem Taktflankensignal. Bei der steigenden (positiven) oder fallenden (negativen) Flanke des Taktes werden die übrigen Eingangssignale ausgewertet und der Ausgang des Flip-Flops gegebenenfalls verändert. Die Flanke, bei der diese Flip-Flops die Eingangssignale auswerten, wird als die aktive Flanke bezeichnet.
Der Takteingang flankengesteuerter Bausteine wird mit einem Dreieck gekennzeichnet. Nachfolgende Abbildung zeigt die Kennzeichnung von Komponenten, welche die positive und die negative Flanke des Takts auswerten. In der linken Komponente ist die positive Flanke des Taktes als aktive Flanke markiert, bei der mittleren und rechten Komponente wird hingegen die negative Flanke als aktive Flanke ausgewählt:



Eine Möglichkeit zur Realisierung eines vorderflankengesteuerten D-FFs ist unten dargestellt. Es handelt sich um den Standardbaustein SN7474. Die dargestellte Schaltung enthält 3 Rückkopplungen. Trennt man diese auf an den Stellen Y1, Y2 und Y3, so entsteht ein zyklenfreies Verknüpfungsnetz.



Vorderflankengesteuertes D-FlipFlop: a) Schaltung, b) Schaltungssymbol, c) explizite Darstellung von Verknüfungsnetz und Rückführungen (ohne Preset und Clear)

Die Analyse ergibt:

$Y_3 = \neg y_1 \vee y_2 y_3$

$Y_2 = \neg c \vee Dy_2 \vee \neg y_1$

$Y_1 = \neg c \vee \neg D y_1 \vee y_1 \neg y_2$

Alle drei Normalformen sind unat, d.h. frei von Verknüpfungshazards, wie wir später noch sehen werden.

In der Zustandstabelle ergeben sich 4 stabile Zustände, nämlich (y1,y2,y3) {011,100,111,110}. Für die Ausgänge gilt

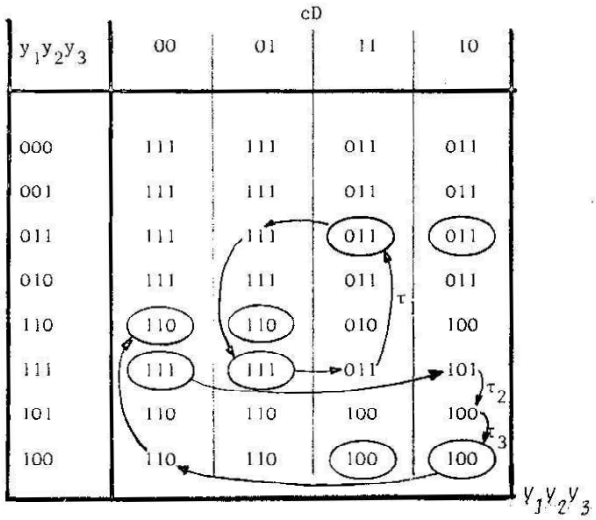
Q = y3

¬Q = ¬(y2y3)

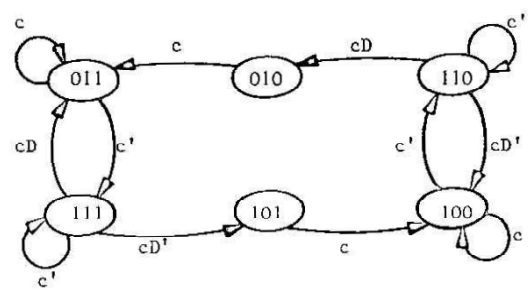
Im Zustandsdiagramm ist erkennbar, daß sich die Schaltung für c=0 stets in einem der Zustände befindet :

Gesetzt: {1,1,1}; Q = 1;
Gelöscht: {1,1,0}; Q = 0;

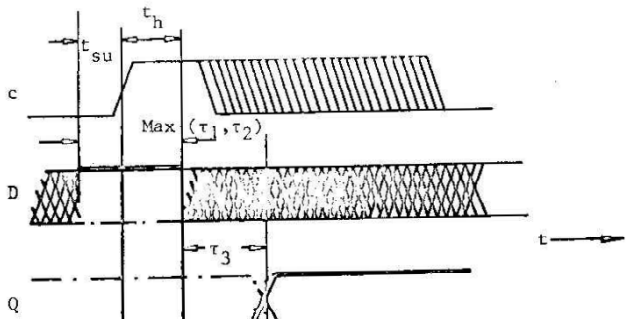
a)



b)



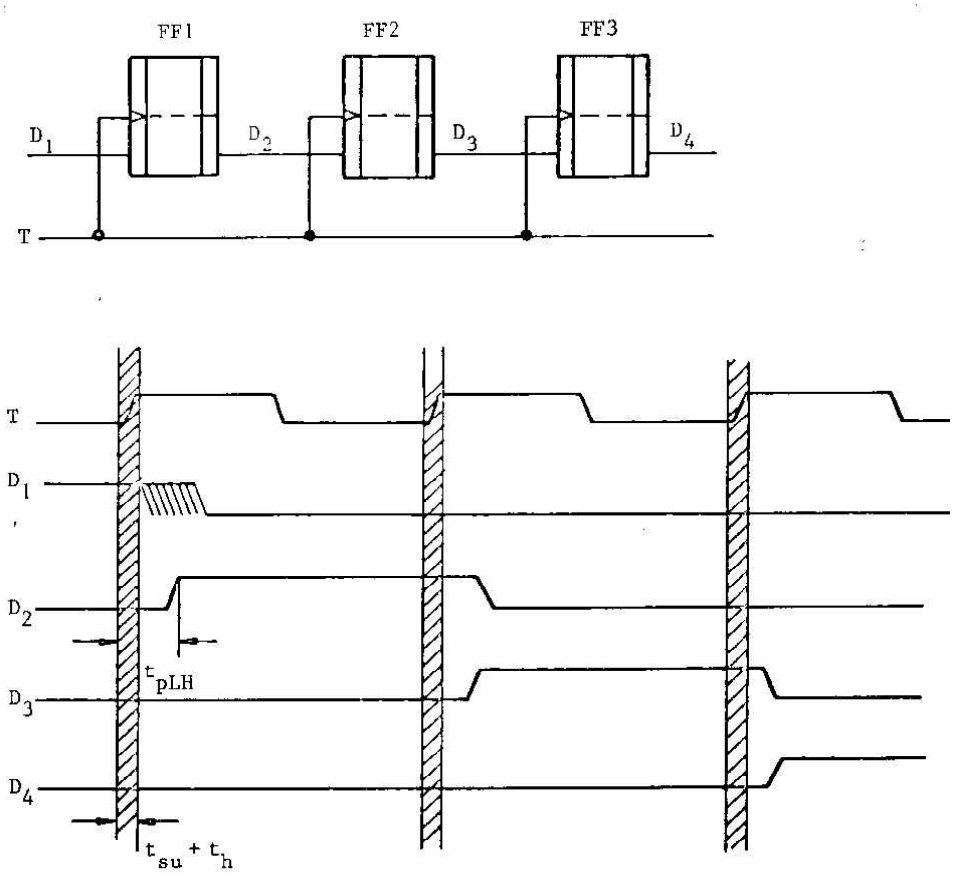
c)



flankengesteuertes D-FF: a) Zustandstabelle, b) Zustandsdiagramm, c) zeitliches Übergangsverhalten

Sobald z.B.ausgehend vom gesetzten Zustand eine steigende Flanke c = 0->1 auftritt, geht die Schaltung bei D=0 in den instabilen Internzustand 101 über, nach kurzer Zeit τ2 in den instabilen Zustand 100 und erreicht dann unabhängig von D nach τ3 den stabilen Zustand

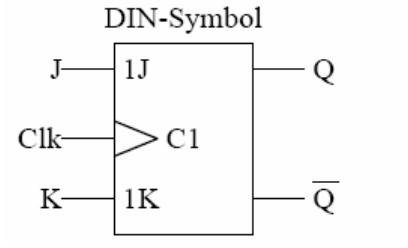
100, siehe Zustandstabelle. Dabei ist Q=y3 von 1 auf 0 gegangen. Mit der abfallenden Flanke geht die Schaltung in den stabilen Zustand 110 über, Q bleibt auf 0.
Ist dagegen D=1, dann geht die Schaltung direkt in den stabilen Zustand 011 über und verharrt dort unabhängig von D, solange c=1 gilt und kehrt mit der fallenden Flanke von c in den Anfangszustand 111 zurück. Während dieser Zeit bleibt Q=1.
Die Funktion der beiden, durch die Gatter 1,2,3,4 gebildeten Hilfs-FFs ist also, nach einer ansteigenden Flanke von c den Dateneingang D zu blockieren, während das Haupt-FF der Gatter 5,6 eingestellt wird. Alle möglichen Übergänge sind Einkomponentenübergänge, also frei von Läufen. Man erkennt am zeitlichen Übergangsverhalten, daß der Wechsel des Ausgangssignals mit Sicherheit nach dem Entscheidungsintervall des Eingangs liegt. Damit können solche FFs direkt verkettet werden, z.B. für die Schaltung als Schieberegister oder Frequenzteiler.



10.1.1.4 JK-Flip-Flop

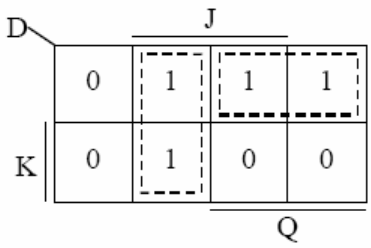
Das JK-Flip-Flop besitzt die Steuereingänge J und K. Die Bezeichnung der Eingänge stammt vom Englischen Jump und Kill.
Dieses Flip-Flop arbeitet wie das RS-Flip-Flop wenn man das Signal J mit S und das Signal K mit R gleichsetzt. Beim JK-Flip-Flop ist jedoch die Kombination J,K=11 spezifiziert, mit dieser Eingangskombination wechselt der Ausgang bei jeder aktiven Taktflanke (für den Fall der Flankensteuerung) seinen Zustand. Man erhält somit die folgende Wahrheitstabelle:

J	K	Clk	Q
-	-	0	Q^*
0	0	↑	Q^*
0	1	↑	0
1	0	↑	1
1	1	↑	$\neg Q^*$
-	-	1	Q^*



Eine Realisierung unter Verwendung eines D-Flip-Flops in Analogie zum vorgestellten Entwurf des flankengesteuerten RS-Flip-Flops, bildet die Eingänge J und K sowie den Ausgang Q auf den D-Eingang eines eingebetteten D-Flip-Flops ab:

J	K	Q	D
0	0	0	0
0	0	1	1
0	1	-	0
1	0	-	1
1	1	0	1
1	1	1	0



Somit ergibt sich die Ansteuerfunktion für den D-Eingang zu:

$D = (J \wedge (\neg Q)) \vee ((\neg K) \wedge Q)$

10.1.1.5 T-Flip-Flop

Das T-Flip-Flop ermöglicht, gesteuert von einem Eingang T, das Halten oder das Wechseln des Ausgangs Q. Mit T=0 wird der Ausgang gehalten, mit T=1 wechselt der Ausgang mit der aktiven Taktflanke. Der Name des Flip-Flops stammt von der englischen Bezeichnung „toggle“. Folgende Wahrheitstabelle faßt dieses Verhalten zusammen:

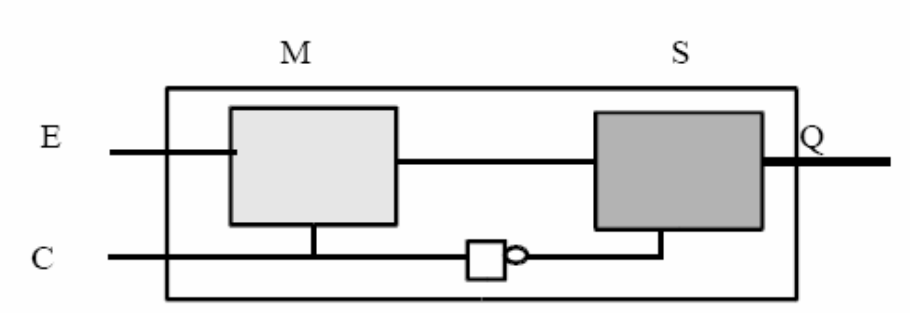
T	Clk	Q
-	0	Q^*
0	↑	Q^*
1	↑	$\neg Q^*$
-	1	Q^*

Die einfachste Realisierung des T-Flip-Flops erfolgt unter Verwendung des JK-Flip-Flops. Verbindet man dort die Eingänge J und K, erhält man direkt ein T-Flip-Flop.

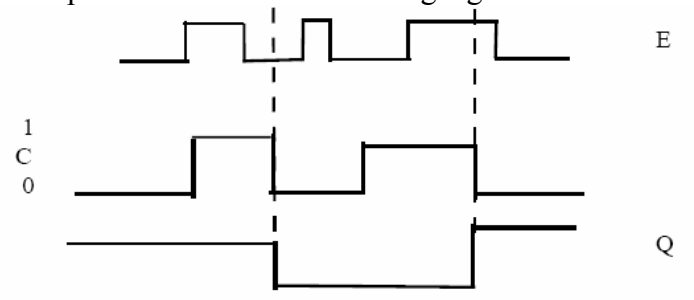
10.1.1.6 Master Slave Flip-Flops

Ein eigentliches Flip-Flop (nichttransparentes Flip-Flop) unterscheidet sich von einem Latch dadurch, daß sich der Zeitpunkt der Übernahme der Eingangswerte von dem des Ausgangsübergangs unterscheidet. Die wesentlichen Eigenschaften eines nichttransparenten Flip-Flops werden durch zwei Zeitintervalle erfaßt. Das eine Intervall gibt die Zeit an, während der das Eingangssignal "gemessen" wird, d.h. während der Eingangssignal für die durch das Taktsignal ausgelöste Zustandsänderung im Flip-Flop relevant ist und deshalb konstant anliegen sollte. Das andere Intervall gibt die Zeit an, während der sich eine Zustandsänderung im Flip-Flop als Binärübergang des Ausgangssignals bemerkbar machen kann. Beide Intervalle dürfen sich nicht überlappen, will man solche Flip-Flops als

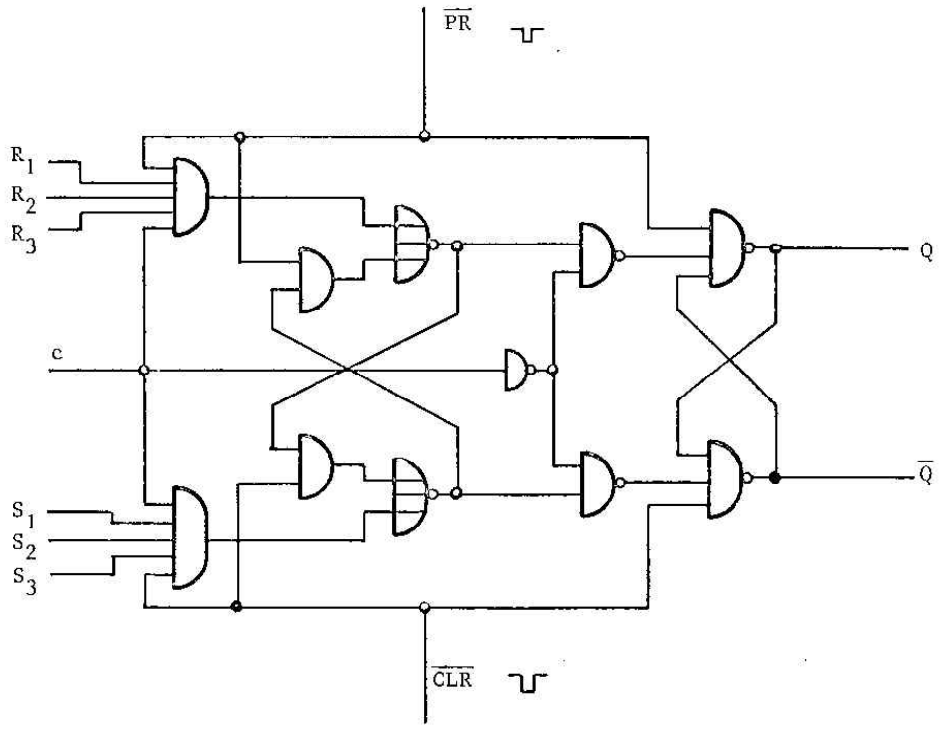
Speicherelemente für ein Schaltwerk verwenden. Dies Verhalten läßt sich durch das Master-Slave-Prinzip (Vorspeicher-Verhalten) erreichen.



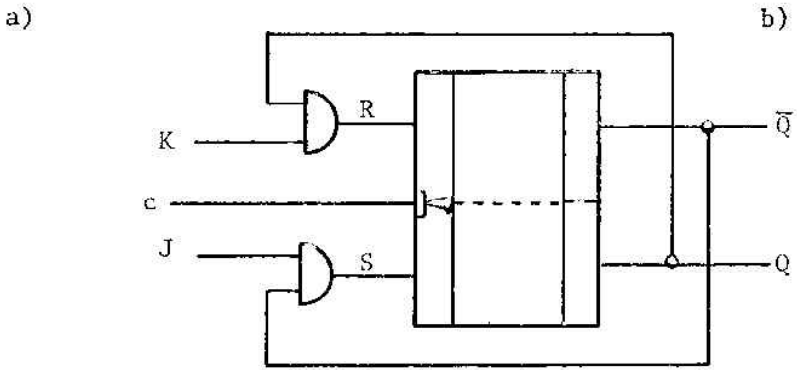
Bei C=1 kann das D-Latch M (Master) den Eingangswert E übernehmen, während sich der Zustand des D-Latch S (Slave) nicht ändern kann, da sein Steuereingang mit 0 belegt ist. Und umgekehrt, bei C= 0 kann sich der Zustand von M nicht ändern, während sich jetzt der Zustand von S ändern kann. Er ändert sich, wenn sich der Zustand von M zuvor geändert hat. Legt man nun an C ein Taktsignal an, dann mißt das Speicherelement bei C = 1
und speichert bei C = 0. Am Ausgang scheint nur die fallende Taktflanke wirksam zu sein.



Das nächste Bild zeigt ein Vorspeicher-RS-FF SN 74L71, Der Vorspeicher ist als NOR-FF und der Folgespeicher als NAND-FF ausgeführt.Außerdem gibt es mehrere Reset- und Set- sowie Preset- und Clear-Eingänge.

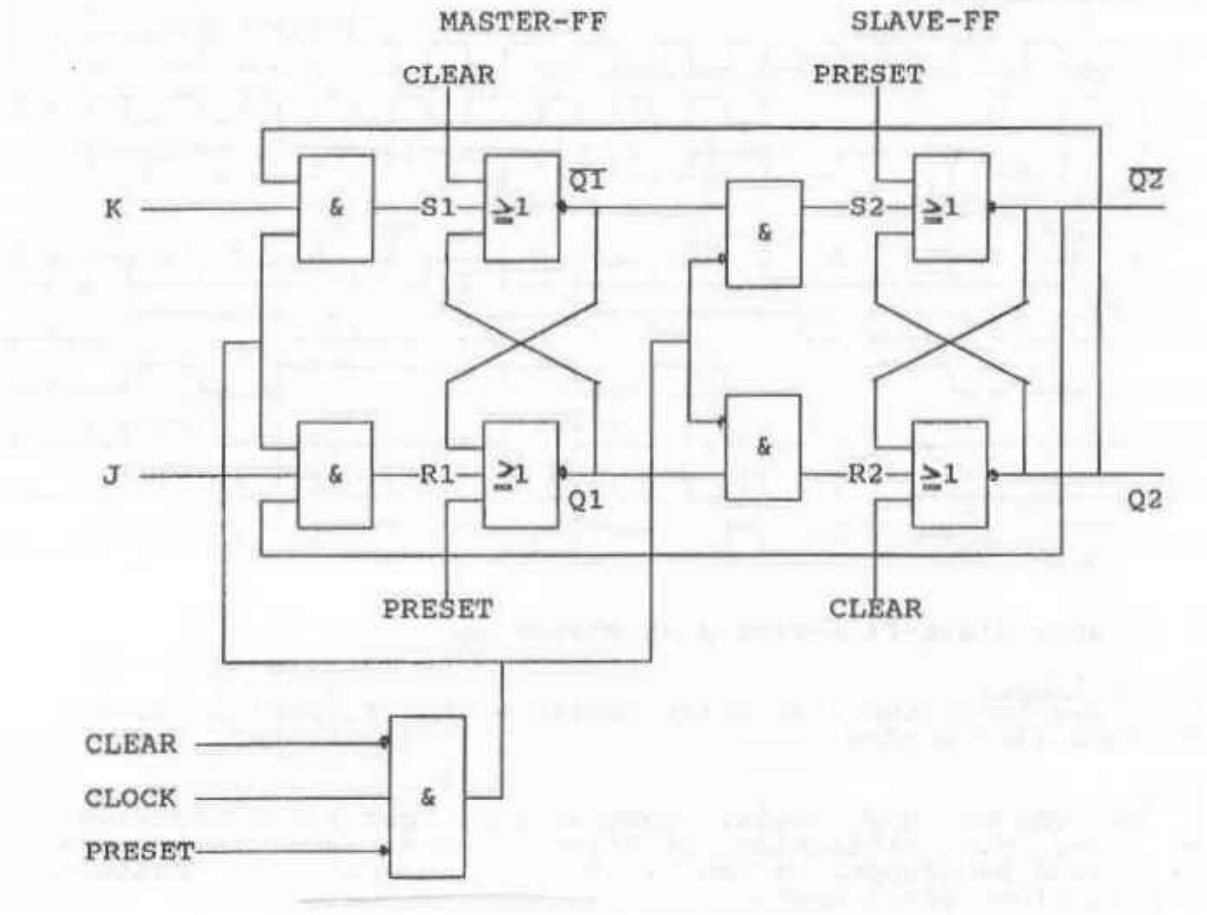


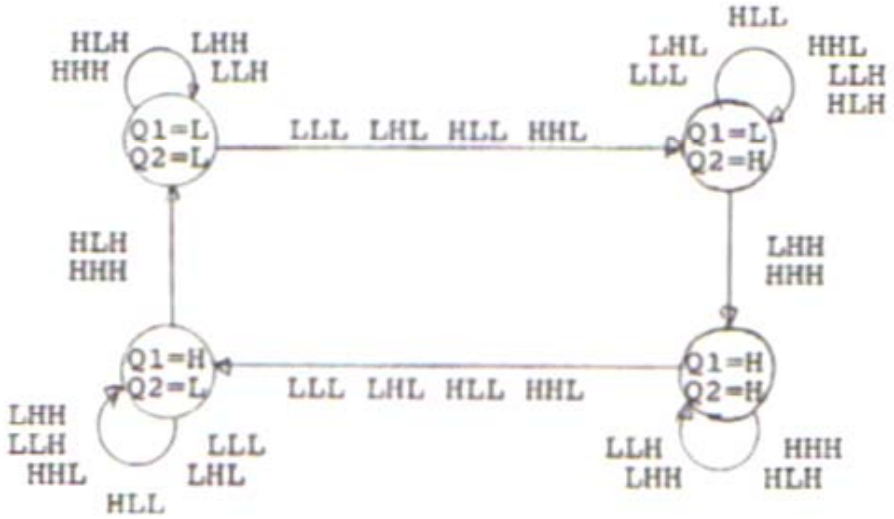
Vorspeicher-RS-FF SN 74L71



J K	Betr.zust.
0 0	speichern
0 1	löschen
1 0	setzen
1 1	wechseln

Konstruktion eines taktpegelgesteuerten JK-Vorspeicher-FFs aus einem RS-Vorspeicher-FF, a) Schaltung, b) Zustände





J,K,clock

<u>Setzen:</u>
Clear=L, ¬Q1=H, ¬Q2=L, Q2=H

J, K, clock haben keine Wirkung, solange Clear=L und Preset=H ist

Rücksetzen:
Clear=H, Q1=H, Q2=L, ¬Q2=H
J, K, clock haben keine Wirkung, solange Clear=H und Preset=L sind

Getakteter Betrieb:
Clear =L, Preset =L

Jetzt gelangen die Clock-Impulse über das NAND-Gatter zu den Eingängen des FFs. Das Verhalten der Schaltung hängt jetzt von J und K ab.
Jeder vollständige Clock-Zyklus besteht aus einer H-Phase und einer daran anschließenden L-Phase.

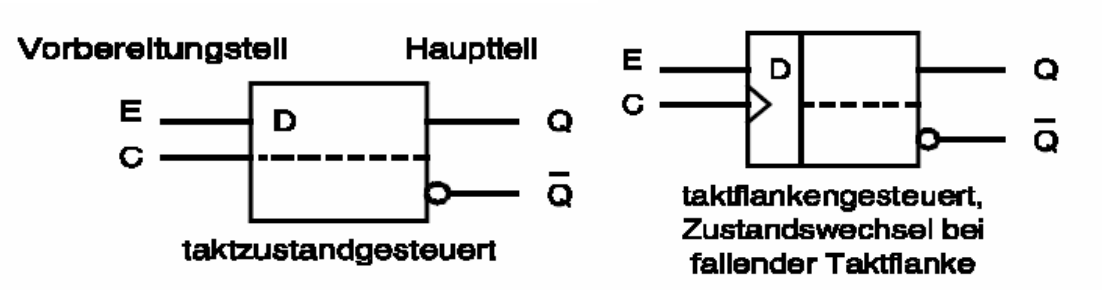
Während der H-Phase schaltet der Masterteil in den durch S1 und R1 festgelegten Zustand. Q1 und ¬Q1 werden dabei ggf. umgesetzt.
Der Slave-Teil bleibt dabei unverändert.

In der anschließenden L-Phase bleibt der Masterteil unverändert, auch wenn die Eingangssignale wechseln sollten. Der Slave-Teil dagegen schaltet jetzt in den durch S2 und R2 festgelegten Zustand, der damit an Q2 und ¬Q2 erscheint.

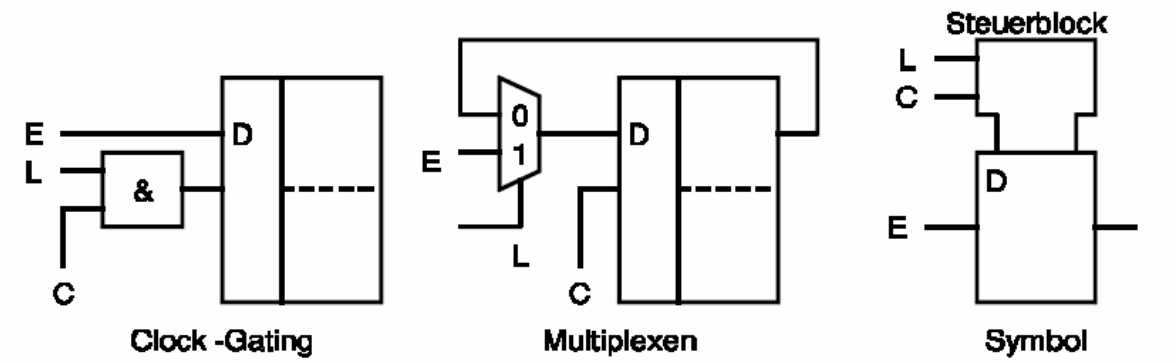
Das Master-Slave-JK-Flip-Flop besitzt 4 interne stabile Zustände, wobei jeweils 2 äußerlich nicht unterscheidbar sind, da Q1 von außen nicht zugänglich ist.

c) Details eines taktpegelgesteuertes JK-Master/Slave-FFs

Zusammenfassung:
Wie gezeigt, besteht in Schaltwerken eine Rückkopplung von den Aus- zu den Eingängen des Speicherteils. Um definierte Verhältnisse zu haben, muß deshalb die Möglichkeit gegeben sein, diese Rückkopplung immer dann aufzubrechen, wenn der Inhalt des Speichers verändert werden soll. Diese Möglichkeit bietet das Master-Slave-Prinzip.



Bei einem synchronen Schaltwerk werden die Steuereingänge der Flip-Flops von einem gemeinsamen Taktgeber, z.B. der Uhr, getrieben. Es ist dann aber erforderlich hin und wieder für bestimmte Flip-Flops einen Zustandswechsel unterdrücken zu können (Taktausblendung). Gezeigt sind zwei Schaltungen, die dies erlauben, vorausgesetzt, daß sich das Ladesignal nur dann ändert, wenn C = 0 ist.

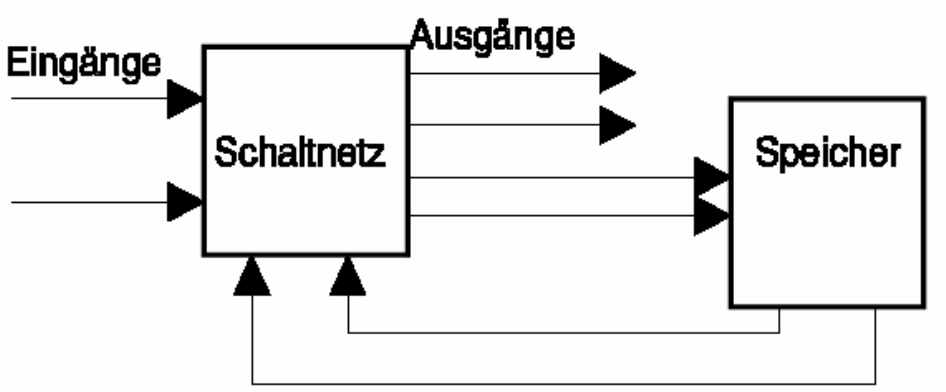

E
L
&
D
C
Clock -Gating
0
1
E
D
L
C
Multiplexen
Steuerblock
L
C
D
E
Symbol

10.2 Synchrone-, asynchrone-, self-timed quasi-synchrone Schaltwerke

10.2.1.1 Asynchrone Schaltwerke

haben Speicherelemente, die zu unterschiedlichen Zeitpunkten Informationen übernehmen. Meist werden die dazugehörigen Takte selbst durch Schaltnetze gebildet, die mit unterschiedlichen Bedingungen und Laufzeiten Clock-Signale erzeugen. Die Speicherelemente können auch implizit durch verteilte Rückkopplungen erzeugt werden und müssen nicht als Flip-Flops erkennbar sein.
Das Verhalten realer komplexer Schaltwerke mit beliebig verteilten Rückkopplungen ist aber kaum vorherzusagen, da es vom Zeitverhalten der Signale und Schaltelemente abhängt. (Klar, S. 99) "Den gesamten zeitlichen Signallauf durch ein größeres Schaltwerk zu verfolgen und zu beschreiben, ist ein so schwieriges Unterfangen, daß sich die sogenannte Synchron-Technik durchgesetzt hat.
Der Teil, der die Eingangswerte nur verarbeitet und nicht speichert, ist ein Schaltnetz. Der speichernde Teil wird aus einfachen Schaltwerken – den Flip-Flops – aufgebaut, die i.a. nur zwei stabile Zustände besitzen. Wird die Speicherfunktion durch einen zentralen Takt gesteuert (d.h. sind die Takteingänge aller Flip-Flops mit einem einzigen Taktgeber (Clock) verbunden), erhält man ein synchrones Schaltwerk; andernfalls erhält man ein asynchrones Schaltwerk. Für ein einwandfreies Funktionieren, müssen die Speichereingänge solange stabil bleiben, bis sich nach Durchlaufen des Schaltnetzes die neuen Signalwerte eingestellt haben.

10.2.1.2 Synchrone Schaltwerke:



1. Jedes Schaltwerk wird als zerlegt in Schaltnetze und Speicherelemente gedacht.
2. Der zeitliche Ablauf der Signale in den Schaltnetzen wird nicht betrachtet, nur das Maximum tsmax der Signallaufzeit von den Eingangsvariablen bis zur Ausgangsfunktion.
3. die Zeitskala wird in diskrete Taktzeiten eingeteilt, so daß zwei Taktzeitpunkte ti wenigstens den Abstand tsmax haben.
4. Jeweils zu den Taktzeitpunkten werden die in den Schaltnetzen asynchron, d.h. ungetaktet gebildeten Informationen in Speicherelemente übernommen.
5. Die neuen Speicherinhalte (innere Zustände des Schaltwerks) werden erst in der nächsten Taktzeit wirksam.“

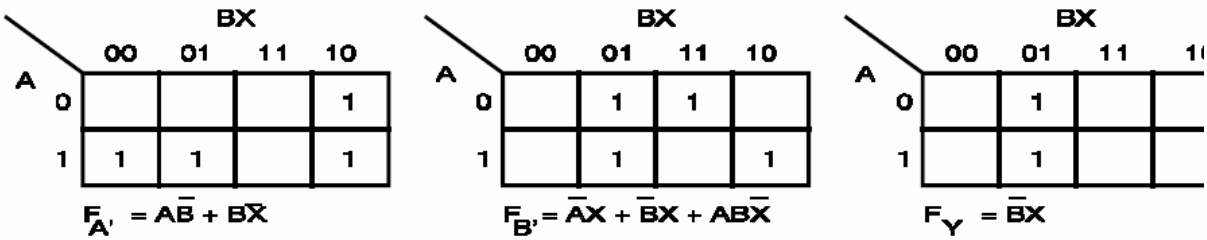
10.2.1.3 Self-timed Schaltwerke:

Die Daten werden durch Verarbeitungsnetze geschickt, die eine bestimmte Laufzeit haben. Der Takt wird ebenso durch gleiche Laufzeiten übertragen, so daß Daten und Laufzeit wieder zusammenpassen. Dieses Prinzip hat Vorteile bei größeren Netzen, da hier nicht alle Register und Treiber gleichzeitig schalten müssen und so die Belastung der Versorgungsspannung besser verteilt wird. Auch EMV Themen sind damit besser zu lösen.

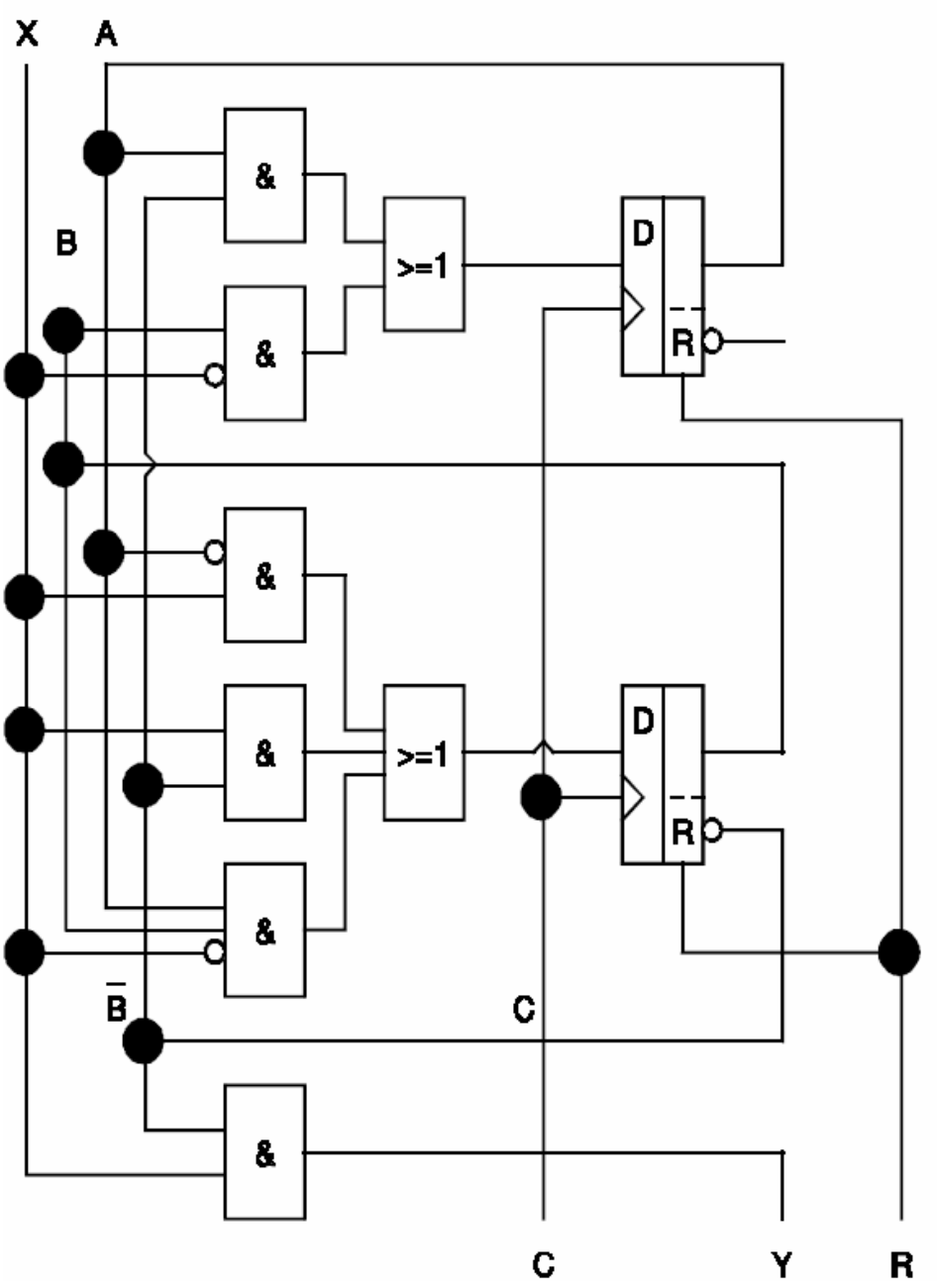
10.2.1.4 Schaltwerk-Synthese

An einem Beispiel soll nun die logische Synthese eines Schaltwerks gezeigt werden. Sie besteht im Grunde darin, den Teil, der das Schaltnetze ausmacht, wie wir es bereits kennengelernt haben, zu entwerfen und dann Flip-Flops hinzuzufügen.
Als erstes legen wir das Verhalten des Schaltwerks, das wir synthetisieren wollen, fest. Dazu beschreiben wir Zustände und abhängige Folgezustände. Aus dieser Verhaltensbeschreibung lassen sich Schaltfunktionen gewinnen, die in Form einer Tabelle dargestellt werden. Dazu müssen wir zusätzlich (zwei) Zustandsvariable einführen. Die Schaltfunktionen werden dann in Schaltnetze überführt. Die Möglichkeit, das Schaltwerk durch ein sogenanntes Reset-Signal R in einen Anfangszustand zu bringen, sei separat berücksichtigt.

Zustand		Eingabe	nächster Zustand		Ausgabe
A	B	X	A'	B'	Y
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	0	0



	00	01	11	10
0	0	1	3	2
1	4	5	7	6

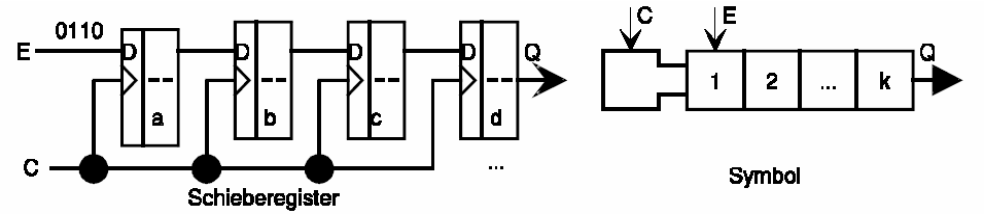


11 Spezielle Schaltwerke

Im folgenden sollen einige Beispiele für Standard-Schaltwerke vorgestellt werden, wie man sie in einem Rechner wiederfindet.

11.1.1 Schieberegister

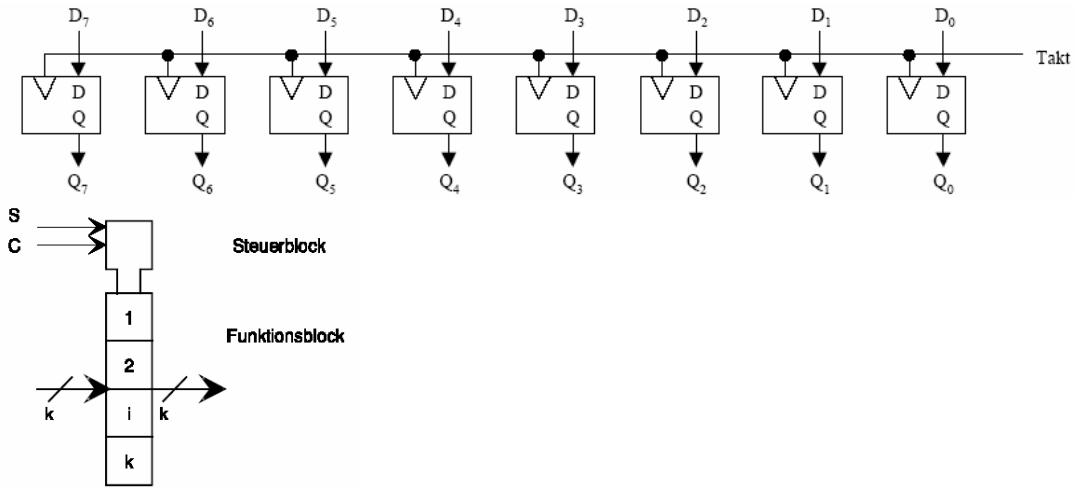
Durch Hintereinander-Schalten (serielle Verknüpfung) von k Flip-Flops erhalten wir ein sogenanntes Schieberegister. Ein solches Schieberegister kann eine Bitfolge der Länge k speichern, indem diese Folge Bit für Bit eingelesen wird. Es kann diese Folge dann auch Bit für Bit wieder ausgelesen werden.
Synchrones Schieberegister mit k = 4.



Dasjenige Bit der Eingabe (E), das am weitesten rechts steht, wird als erstes eingegeben. Wenn abcd der anfangs gespeicherte Inhalt dieses Schieberegisters ist und die Bitfolge 0110 taktweise eingelesen wird, erscheint am Ausgang zunächst Takt für Takt erst d dann c dann b dann a und erst in den nächsten 4 Takten erscheint die Bitfolge 0110. (Das rechte Bit wird als erstes ausgegeben).

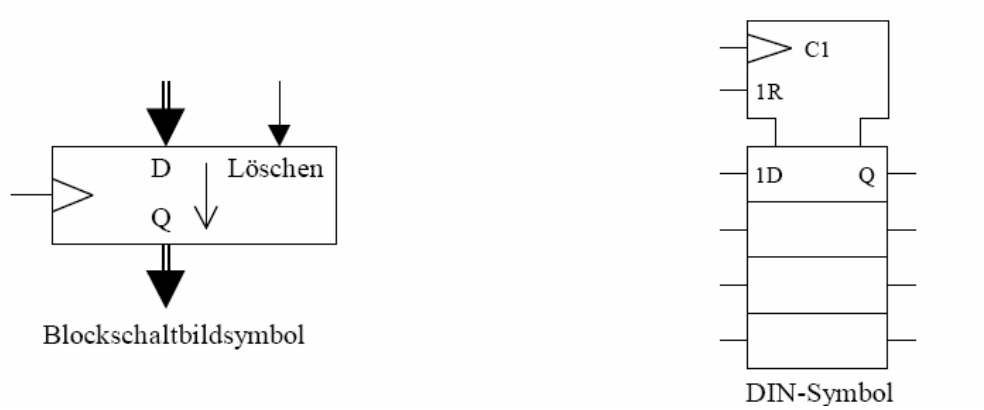
11.1.2 Register

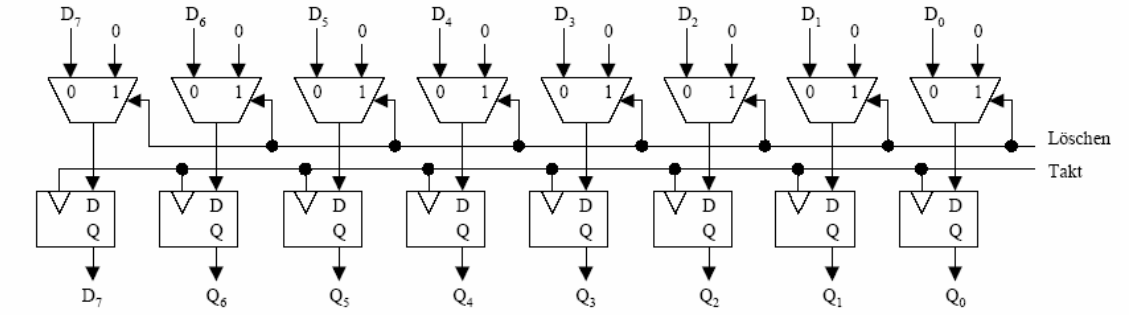
Um mehrere Bits gleichzeitig speichern zu können, kann man n Flip-Flops parallel verschalten. Diese Verschaltung liefert ein Register der Breite n. In einem solchen Register kann ein n-Bit Wort mit einem Taktsignal gespeichert werden (PIPO). Lesen (Ausgabe) des Registerinhalts ist immer möglich.



11.1.3 Synchron löschbares Register

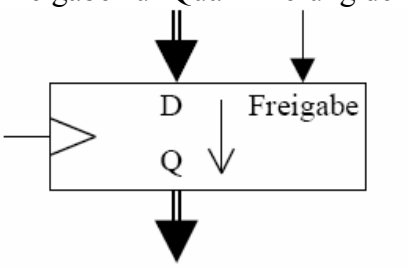
Bei einem synchron löschbaren Register kann über einen Steuereingang Löschen ausgewählt werden, ob bei der nächsten aktiven Taktflanke die Eingangssignale D oder die Werte 0 in die Flip-Flops übernommen werden. Nachfolgende Abbildung zeigt ein Blocksschaltbildsymbol und das DIN-Symbol des synchron löschbaren Registers:



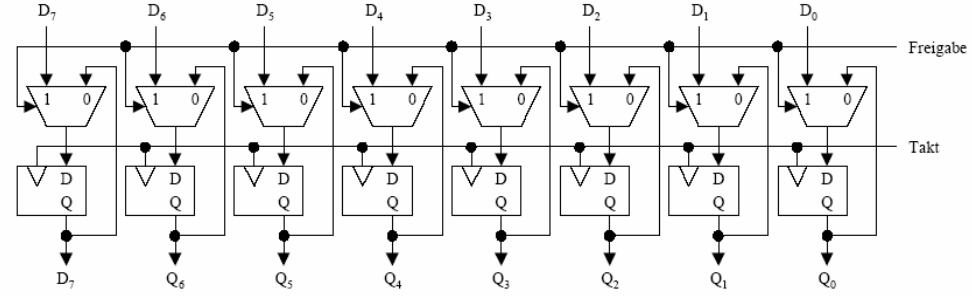


11.1.3.1 Register mit Freigabeeingang

Bei einem Register mit Freigabeeingang kann durch einen zusätzlichen Eingang Freigabe gesteuert werden, ob mit der aktiven Taktflanke das Register die Werte vom D-Eingang übernimmt oder seinen alten Wert behält. Im Englischen wird der Eingang Freigabe üblicherweise mit Enable (Abkürzung EN) bezeichnet. In nachfolgender Abbildung ist ein Blockschaltbildsymbol für das Register gezeigt. Neben dem D-Eingang ist der Steuereingang Freigabe zur Qualifizierung der Übernahme dargestellt.

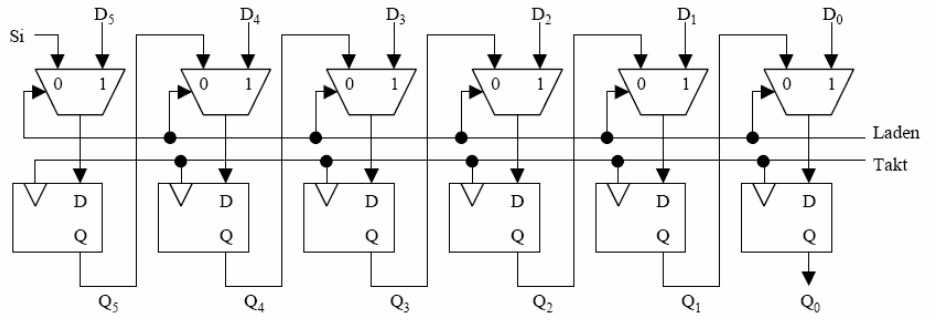


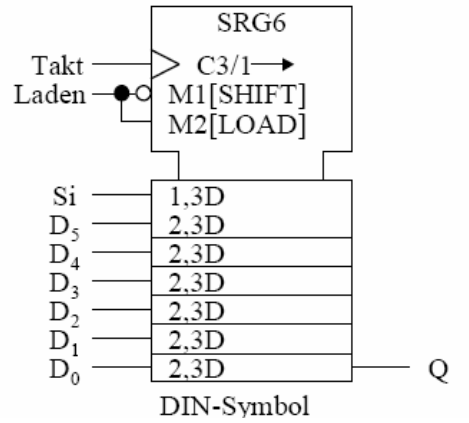
Zur Realisierung der beschriebenen Funktionalität kann man vor jedes Flip-Flop des Registers einen Multiplexer vorschalten:



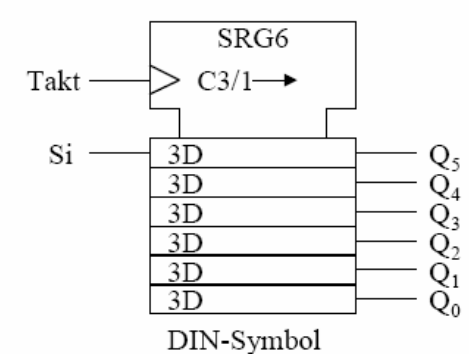
11.1.3.2 Spezielle Schieberegister

Es gibt eine Reihe weiterer Registertypen, das Schieberegister vom Typ PISO mit parallelem Ein- und seriellem Ausgang

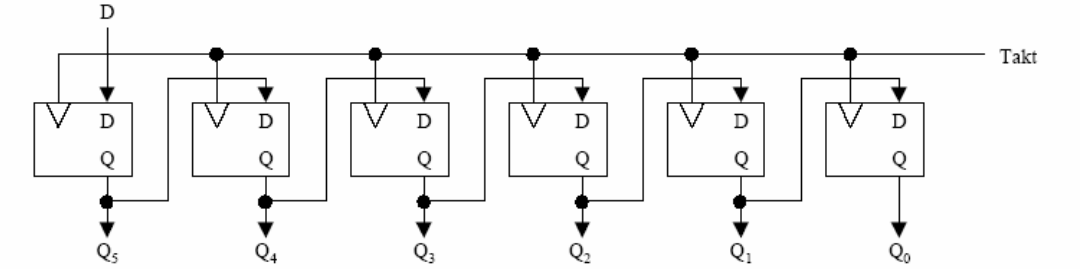




oder umgekehrt SIPO.



Das Rechts-Schieberegister mit parallelem Ausgang ist aus Flip-Flops aufgebaut. Die Struktur ist in nachfolgender Abbildung gezeigt. Am parallelen Ausgang werden bei einem N-Bit Schieberegister dieses Typs immer die letzen N über den D-Eingang eingetakteten Bits ausgegeben.

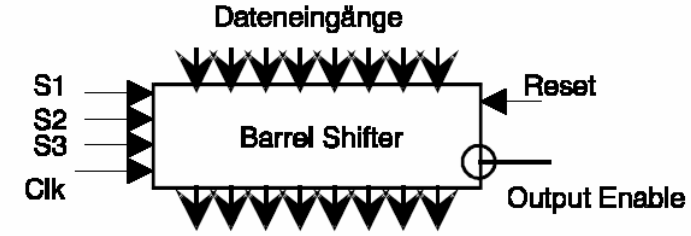


11.1.3.3 FIFOS

Weiterhin gibt es Registertypen mit einer ganzen internen Speicherbank, auf die von zwei Seiten unabhängig zugegriffen werden kann, ohne äußere Synchronisierung der Zugriffe. Das FIFO gibt am Ausgang das zuerst aus, was auf der anderen Seite zuerst eingeschrieben wurde (first in first out).
Das LIFO gibt das am Ausgang zuerst aus, was auf der anderen Seite als letztes eingeschrieben wurde.

11.1.3.4 Shifter

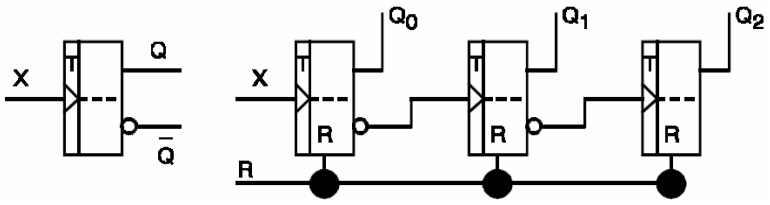
Ein spezialisiertes Register ist der sogenannte Barrel Shifter, der in einem Taktzyklus das shiften (genauer rotieren) um eine beliebigen Anzahl von Bits erlaubt. Diese Anzahl wird über die Steuereingänge Si festgelegt.

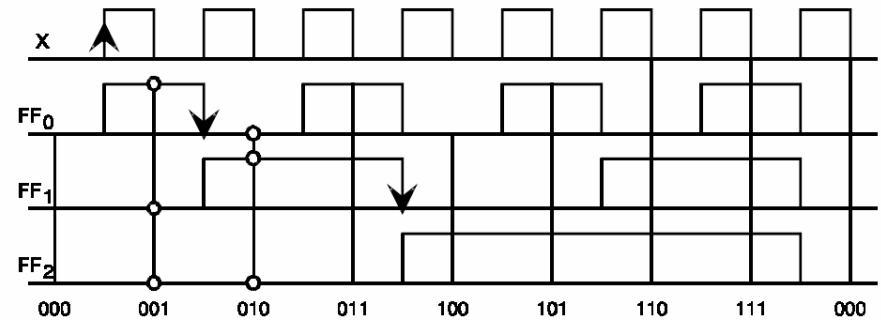


S1S2S3 Steuercode für die Schiebedistanz

11.1.3.5 Asynchroner Zähler mit T-Flip-Flops

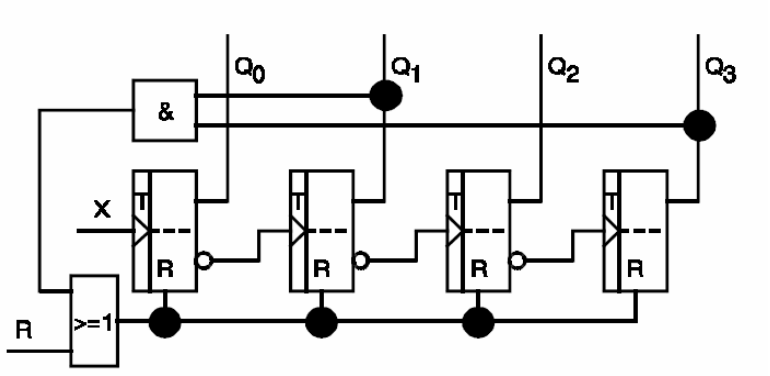
Bei einem T-Flip-Flop (Toggle-FF) ändert mit jeder ansteigenden Flanke des Eingangssignals seinen Zustand. Werden solche Flip-Flops über den negativen Ausgang geschaltet, erhält man einen asynchronen Binärzähler. R ist ein Reset-Eingang zum Zurücksetzen des Zählers auf 0.





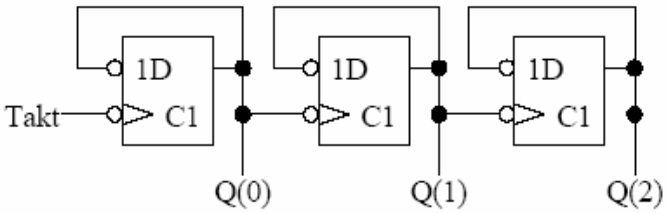
Der vorgestellte Zähler ist ein asynchroner serieller Zähler. Der vom Eingangstakt ausgelöste Zustandswechsel am ersten Flip-Flop pflanzt sich von Flip-Flop zu Flip-Flop fort (ripple counter). Das Zählen beruht hier auf dem Prinzip der Frequenzteilung.

Wir können einen Zähler (modulo 16) verwenden, um ihn zu einem BCD-Zähler auszubauen. Ausgang $Q_3Q_2Q_1Q_0 = 1010_{(2)} = 10_{(10)}$ setzt den Zähler zurück (R: reset). Bei aktivem R-Eingang gehen die Flip-Flops in den Zustand 0 über.



11.1.3.6 Einfacher asynchroner Zähler mit D-Flip-Flops

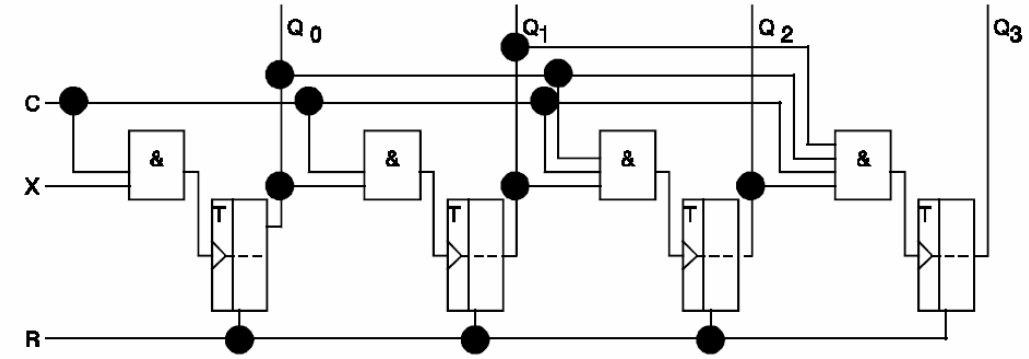
Beim asynchronen Zähler verbindet man den Ausgang des vorhergehenden Flip-Flops mit dem Takteingang des nachfolgenden Flip-Flops. Dann wird das nachfolgende Flip-Flop mit der halben Frequenz seines Vorgängers getaktet und wechselt somit auch nur halb so häufig seinen Zustandswert. Ein 3-Bit asynchroner Zähler kann somit mit folgender Schaltung



realisiert werden:
Man beachte die Verzögerung, die sich von Ausgang zu Ausgang ergibt. Sie resultiert daher, daß ein Ausgang als Taktsignal der nächsten Zählerstufe verwendet wird. Durch diese Verzögerungen wird am Ausgang nun nicht sauber hochgezählt, sondern an den Übergängen ergeben sich kurzzeitig falsche Zählerwerte.

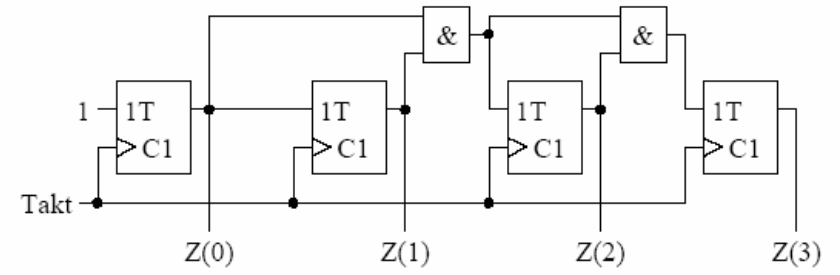
11.1.3.7 Einfacher synchroner Zähler mit T-Flip-Flops

Bei einem synchronen Zähler wird ein Zählimpuls allen Flip-Flops gleichzeitig zugeführt. Ein derartiger Zähler ist aufwendiger als ein asynchroner Zähler. Er ist aber weniger anfällig gegenüber Störungen im Zeitverhalten, da der Ausgangszustand nur während eines kurzen Zeitintervalls nach dem Takt nicht definiert ist.



Achtung: das Bild hat Hazard Probleme, so geht es nicht!

So ist das richtig:



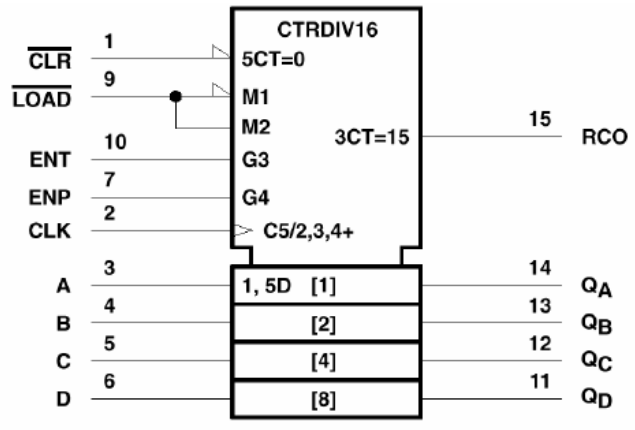
$T(0) = 1$
$T(1) = Z(0)$
$T(2) = Z(0) \wedge Z(1) = T(1) \wedge Z(1)$
$T(3) = Z(0) \wedge Z(1) \wedge Z(2) = T(2) \wedge Z(2)$

11.1.3.8 Synchrone Zähler mit Steuereingängen und -ausgängen

Dualzähler können Freigabe- und Rücksetzeingänge besitzen, weiterhin können sie ladbar ausgeführt werden:

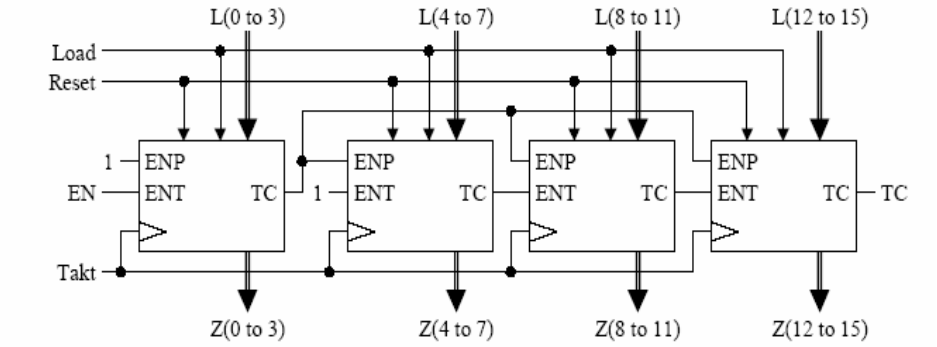
- Besitzt ein Zähler einen Freigabeeingang, muß das angeschlossene Signal während der aktiven Taktflanke seinen aktiven Wert annehmen, damit der Zähler zählt. Ansonsten hält der Zähler seinen augenblicklichen Wert.
- Bei Rücksetzsignalen unterscheidet man zwischen synchronen und asynchronen Signalen. Ein aktives asynchrones Rücksetzsignal bewirkt, daß der Zähler sofort auf den Wert 0 gesetzt wird, ohne daß die Taktflanke berücksichtigt wird. Ein aktives synchrones Rücksetzsignal führt dazu, daß der Zähler mit der nächsten aktiven Taktflanke auf den Wert 0 zurückgesetzt wird.
- Bei einem ladbaren Zähler bewirkt ein aktives Ladesignal, daß der Wert eines parallelen Datenvektors mit der nächsten aktiven Taktflanke als Zählerstand geladen wird.

Neben dem Ausgangsvektor mit dem aktuellen Zählerstand findet man bei Aufwärts-Dualzählern ein Ausgangssignal, welches mit „Terminal Count“ bezeichnet ist. Dieses Signal markiert, daß der Zähler auf dem höchsten Zählerstand steht. Dieses Signal dient beim Kaskadieren mehrerer Zähler als Freigabesignal der nächst höheren Zählerstufe.



11.1.3.8.1 Kaskadieren von Zählern

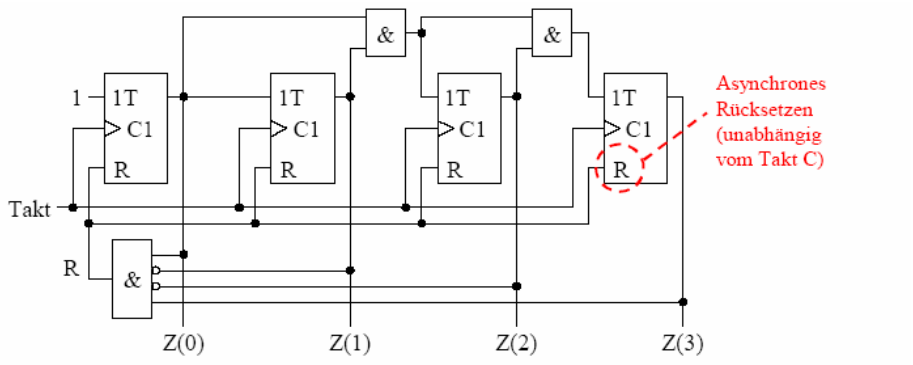
Mehrere Zählerbausteine lassen sich kaskadieren, um Zähler mit größerer Wortbreite zu erhalten. Die beiden
Freigabesignale ENP und ENT des Zählers aus dem vorhergehenden Abschnitt dienen direkt der Kaskadierung.
Nachfolgendes Blockschaltbild zeigt, wie die Kaskadierung mehrerer Zähler erfolgen muß:



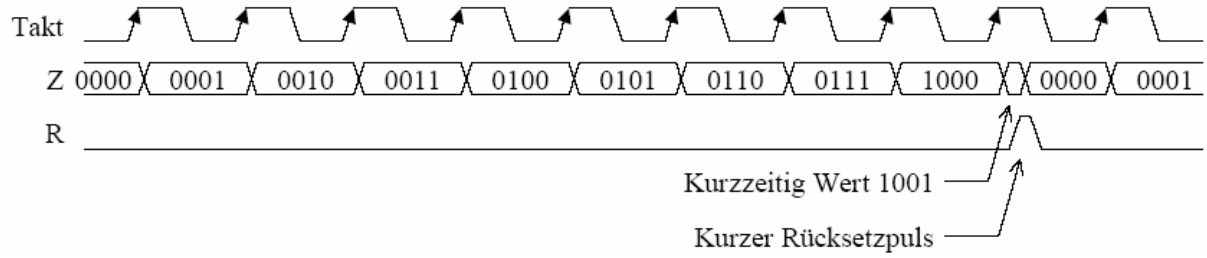
Achtung, normale Kaskadierung führt zu deutlicher Verringerung der maximalen Betriebsfrequenz, der Carry (TC) würde immer kürzer werden.
Der Vorgriff mit TC auf ENT hilft. Nur wenn ENP und ENT 1 sind, ist der Zähler enabled. ENP beeinflußt TC out nicht, d.h. der Carry steht schon früh ab Takt 0 des 15ten Zyklus' an.

11.1.3.9 Modulo-N Zähler

In vielen Anwendungen benötigt man einen Zähler, der eine fest vorgegebene Anzahl von Ereignissen zählt.
Dazu ergänzt man die vorgestellten Binärzähler mit einer Schaltung, welche beim Erreichen der vorgegebenen Anzahl den Zähler zurück auf 0 setzt. Zur Realisierung des Rücksetzens findet man zwei Alternativen, asynchrones und synchrones Rücksetzen.
<u>Asynchrones Rücksetzen</u>
Beim asynchronen Rücksetzen werden asynchrone Rücksetzsignale der Flip-Flops ausgenutzt. Zum Rücksetzen verwendet man ein UND-Gatter über alle Zählerausgänge, welches beim Erreichen des Wertes N ein asynchrones Rücksetzsignal auf die Flip-Flops gibt. Soll beispielsweise ein Modulo-9 Zähler realisiert werden, nimmt man dazu den nächst größeren Dualzähler und schaltet die Zählerausgänge unter Verwendung von Invertern so auf ein UND-Gatter, daß dieses beim Wert Z = 9 = (1001)2 einen Ausgangswert 1 liefert1. Mit dem Ausgang des UND-Gatters setzt man dann die Zähler-Flip-Flops zurück:

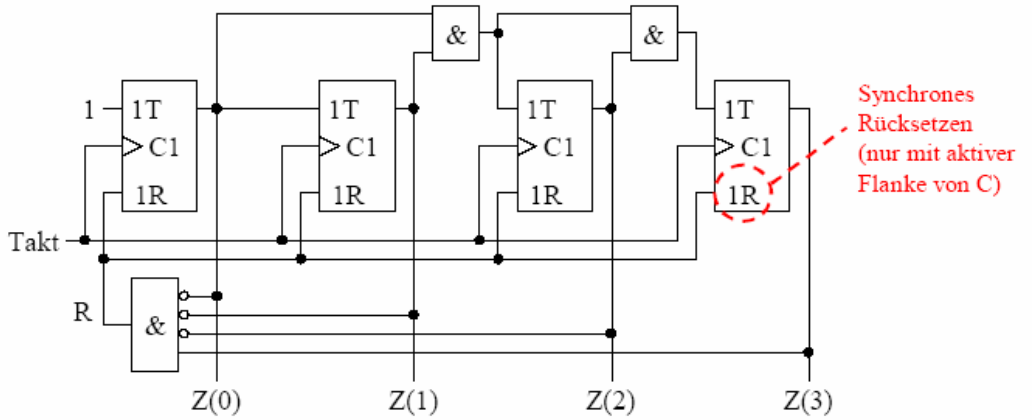


Es reicht aus, nur die Zählerausgänge mit UND zu verknüpfen, die beim Zählende auf 1 liegen. Damit kann im Beispiel zur Generierung des Rücksetzsignals R ein UND-Gatter mit 2 Eingängen verwendet werden, welches die Zählerausgänge Z(0) und Z(3) verknüpft. Problematisch bei diesem Vorgehen ist, daß zum Rücksetzen kurzzeitig der Wert N (im Beispiel N=9) am Ausgang des Zählers anliegt. Auch ist die Breite des Rücksetzpulses durch die Verzögerungszeiten des UNDGatters und der Flip-Flops bestimmt.

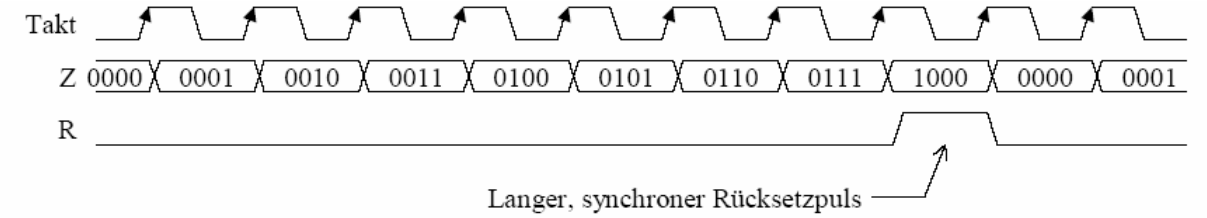


11.1.3.9.1 Synchrones Rücksetzen

Beim synchronen Rücksetzen ermittelt man den Wert N-1 durch ein UND-Gatter und wählt Flip-Flop-Typen mit einem synchronen Rücksetzeingang. Dann erhält man für den beispielhaften Modulo-9 Zähler die folgende Schaltung:



Nun wird der Wert $Z = 8 = (1000)_2$ durch das UND-Gatter erkannt, der bei der nächsten aktiven Taktflanke ein synchrones Rücksetzen der Flip-Flops bewirkt. Der Unterschied tritt am deutlichsten im Zeitdiagramm hervor:



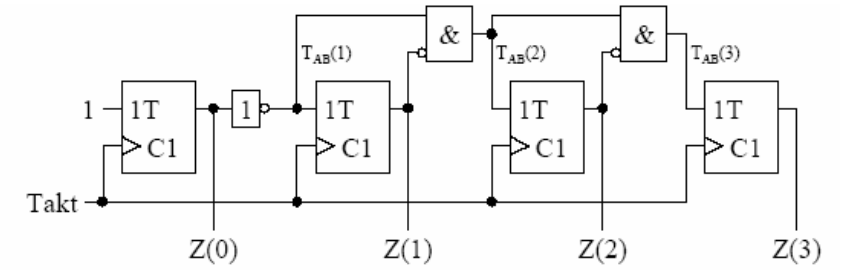
Der Rücksetzpuls wird nun schon beim Zählerstand N=8 generiert, so daß mit der nächsten Taktflanke die Zählerausgänge sofort auf den korrekten Wert 0 geschaltet werden. In dem Fall N=8 reicht auch die Rückkopplung von Z(3) ohne weitere Und-Verknüpfung aus.

11.1.3.10 Auf-/Abwärtszähler

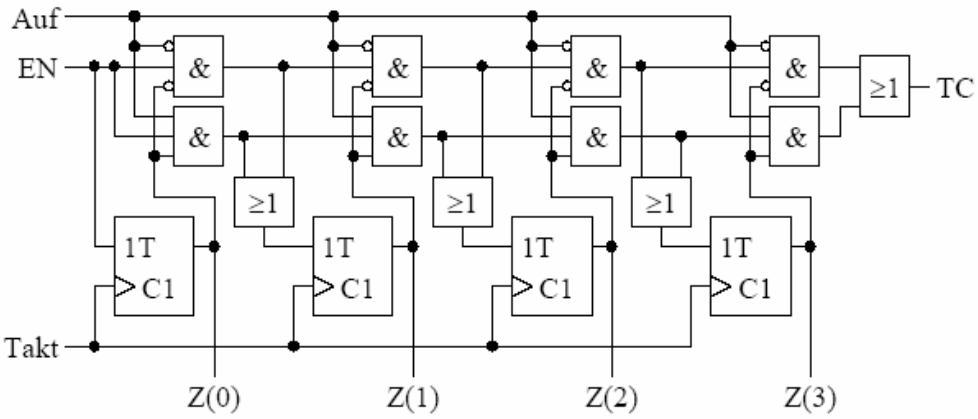
In einem Abwärtszähler wird eine Zählerstelle dann umgeschaltet, wenn alle darunter liegenden Stellen den Wert 0 angenommen haben. Das Umschalten erfolgt dann mit der nächsten aktiven Taktflanke. Es ergeben sich damit folgende Toggle-Bedingungen für Abwärtszählen:

TAB(0) = 1
TAB(1) = ¬Z(0)
TAB(2) = TAB(1) ∧¬Z(1)
TAB(i) = TAB(i-1) ∧¬Z(i-1) für i≥2

Durch Invertierung von Eingängen der UND-Gatter, mit denen die Wechsel-Bedingung der T-Flip-Flops ermittelt wird, kann beispielsweise der zuvor besprochene 4-Bit Aufwärtszähler in einen Abwärtszähler umgewandelt werden:



Kombiniert man den Aufwärts- mit dem Abwärtszähler, so erhält man einen umschaltbaren Auf-/Abwärtszähler, bei dem mittels eines Signals Auf zwischen Auf- und Abwärtszählen gewählt werden kann:

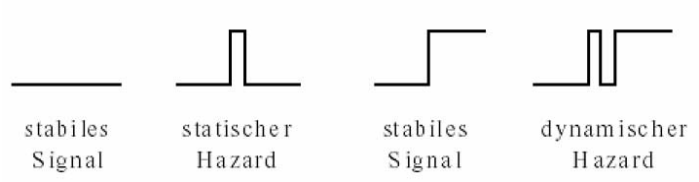


12 Hazards

Hazard (Wagnis, Gefahr) = zeitlich begrenzter Fehler in digitalen Schaltungen (Laufzeiteffekt). Hazards sind kurzzeitige und unerwartete Änderungen eines Wertes auf Signalleitungen, die sich auf die Stabilität digitaler Schaltungen sehr negativ auswirken können. Daher ist es sinnvoll, Hazards möglichst beim Entwurf digitaler Schaltungen zu eliminieren.
Entstehung von Hazards:
• Abhängig von den Laufzeiten in den Gattern werden Signale in den Schaltungen unterschiedlich lange verzögert.
• Theoretisch gleichzeitige Signaländerungen treten real mit einem gewissen zeitlichen Versatz auf

-> Signalwechsel können daher zu unterschiedlichen Zeiten ein Gatter erreichen und ein ungewolltes Schalten bewirken. Der aufgrund von Hazards auftretende Störimpuls wird Hazardimpuls, Glitch oder auch Spike genannt.

Signalverläufe von Hazards



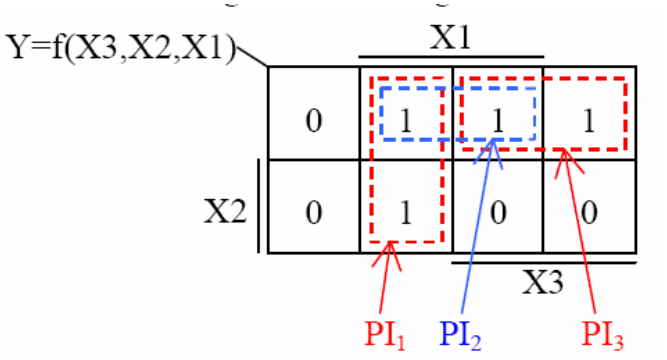
Man unterscheidet bezogen auf die Ursachen zwischen Logik- und Funktionshazards.

12.1.1.1 Logik-Hazards:

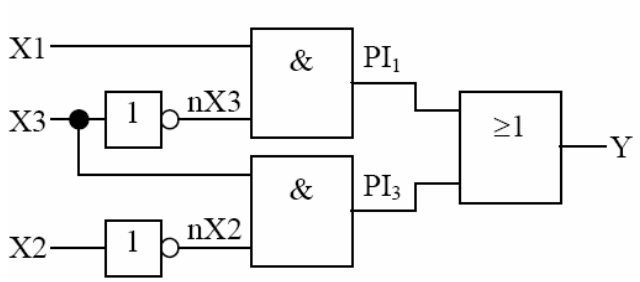
Verursacht durch einen Signalwechsel an einem einzigen Eingang.
Bedingungen für das Auftreten:
• Das vom Eingang kommende Signal verzweigt im Schaltnetz
• Die Signale laufen in einem Gatter wieder zusammen
• Auf den benutzten Signalpfaden treten Laufzeitunterschiede auf

Insbesondere in asynchronen Schaltungsteilen und bei Taktsignalen führen solche Hazards zu fatalen Fehlfunktionen.
Beispiel eines Hazards
Betrachten wir als Beispiel die folgende Funktion:

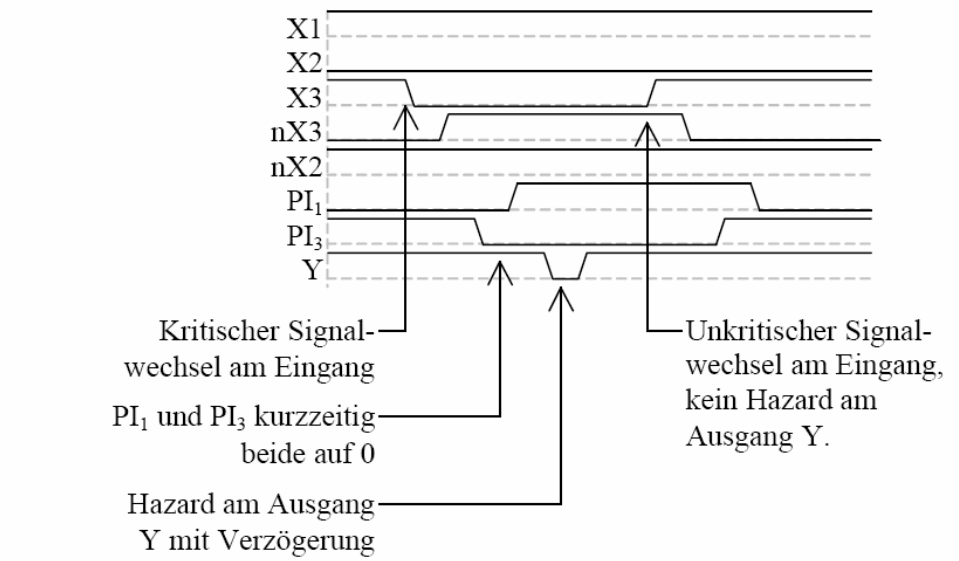
X3	X2	X1	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



Man findet drei konjunktive Primimplikanten PI1, PI2 und PI3, von denen PI1 und PI3 Kern-Primimplikanten
sind. Sie reichen aus, um die disjunktive Minimalform aufzustellen:
Y = PI1 ⩓ PI3 = (X1∧(¬X3)) ⩓((¬X2)¬X3).
Betrachten wir nun den Fall, daß die Eingänge X_3,X_2,X_1 zwischen den Werten 001 und 101 wechseln. Bei beiden Eingangskombinationen liefert die boolesche Gleichung den Wert Y=1. Bei der Eingangskombination 001 wird der 1-Wert jedoch durch den Primimplikanten PI_1, bei der Kombination 101 hingegen durch PI_3 erzeugt. Eine reale Schaltung zur Funktion, die auf der ermittelten disjunktiven Normalform basiert, besitzt die folgende Struktur:



Bauelement	Bezeichnung der Verzögerungszeit	Angenommener Wert der Verzögerungszeit
Inverter	td_{NOT}	2 ns
UND-Gatter	td_{AND}	4 ns
ODER-Gatter	td_{OR}	4 ns



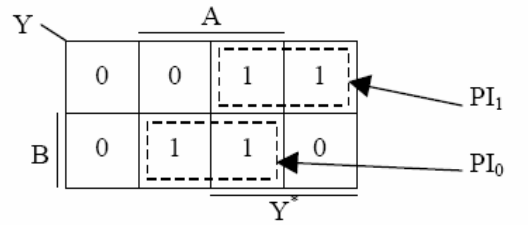
Man erkennt deutlich, daß durch die unterschiedlichen Signallaufzeiten zwischen den Eingängen und Ausgang kurzfristig beide Primimplikanten PI_1 und PI_3 den Wert 0 annehmen. Dies bewirkt, daß am Ausgangssignal ein kleiner Einbruch im Zeitverlauf auftritt, für ca. 2 ns geht der sonst auf 1 liegende Ausgang auf den Wert 0.
Im Beispiel erscheint der Hazard am Ausgang, nachdem der Primimplikant PI3 deaktiviert wurde, der neue Primimplikant PI1 aber noch nicht aktiv ist. Der Hazard erscheint also nach dem kritischen Wechsel von X3 mit der Verzögerung (tdAND+tdOR)=8ns am Ausgang. Der neue Primimplikant PI1 wird erst nach der Verzögerung von (tdNOT+tdAND+tdOR)=10ns aktiv, so daß der Ausgang für die Differenzzeit tdNOT=2ns den Hazard aufweist.
Ob ein Hazard entsteht hängt davon ab, wie schnell eine Änderung durch unterschiedliche Teile der Schaltung weitergegeben wird. Eine Signaländerung propagiert also auf unterschiedlichen Pfaden durch die Schaltung bevor alle Pfade am Ausgang der Schaltung wieder kombiniert werden. Je nach Laufzeit der Änderung durch die unterschiedlichen Pfade entsteht am Ausgang ein Hazard oder nicht. Das Propagieren von Eingangsänderungen durch die Schaltung wird als „Rennen“ (Englisch: „Race“) bezeichnet. Das Auftreten eines Hazard hängt vom Ergebnis dieses Rennens ab.

12.1.1.2 Hazards bei asynchronen Schaltungen

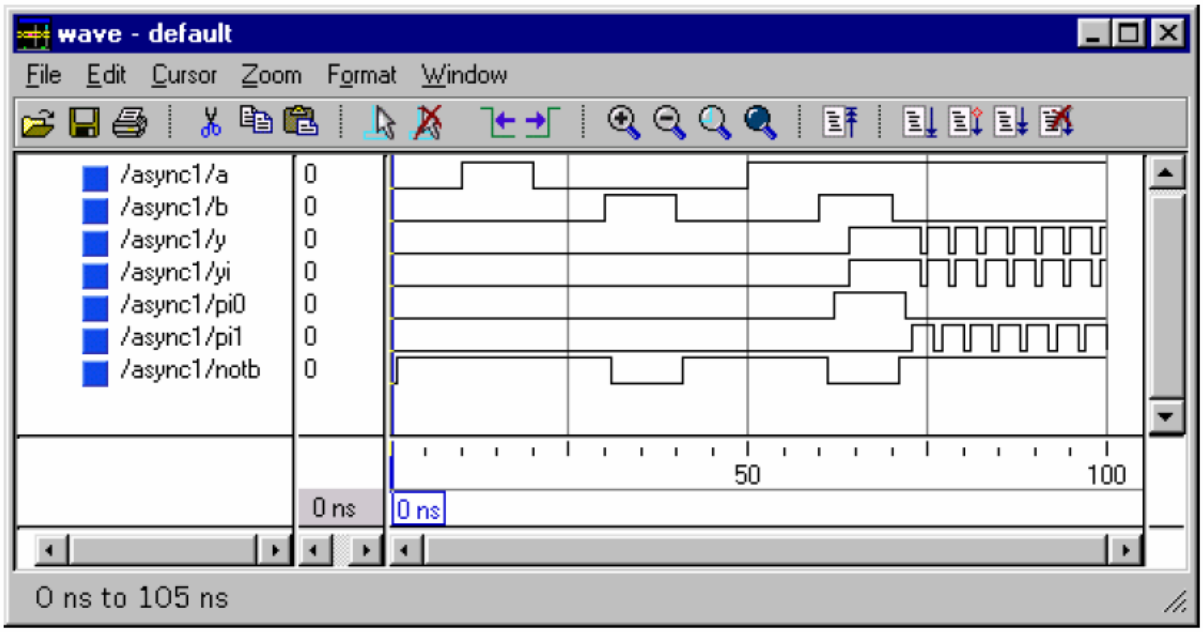
Beim Entwurf asynchroner Schaltungen muß beachtet werden, daß keine Hazards bei der Berechnung des Ausgangssignals auftreten, ansonsten kann eine Schaltung völlig unerwartetes Verhalten zeigen.
Zur Demonstration soll eine Schaltung mit zwei Eingängen A und B und einem Ausgang Y mit folgendem Verhalten entworfen werden:

- Wenn der Ausgang Y=0 ist, soll die Schaltung dann den Ausgang auf 1 setzen, wenn sowohl A=1 als auch B=1 gilt.
- Wenn der Ausgang Y=1 ist, soll die Schaltung dann den Ausgang auf 0 setzen, wenn A=0 und B=1 gilt.

Die Spezifikation führt zu folgendem KV-Diagramm:



Man erkennt die beiden Primimplikanten $PI_0 = A \wedge B$ und $PI_1 = (\neg B) \wedge Y^*$. Somit ergibt sich die disjunktive Minimalform bei gedachter Auftrennung des Rückkopplungspfades zu:
$Y = PI_0 \,\dot{\vee}\, PI_1 = (A \wedge B) \dot{\vee} ((\neg B) \wedge Y^*)$.

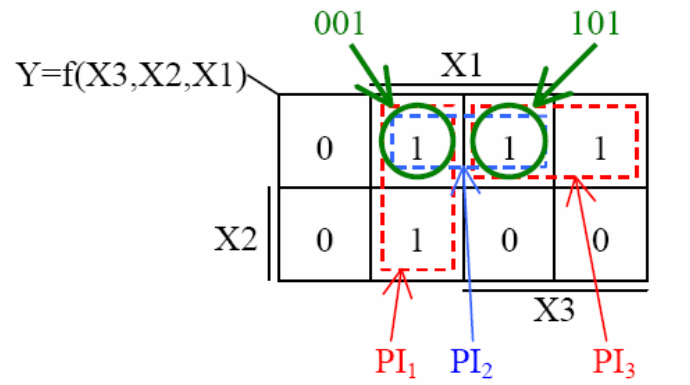


Zunächst wird bei Y=0 das Signal A allein auf 1 gesetzt und der Ausgang verharrt im Zustand 0. Ebenso verharrt der Ausgang auf 0, wenn allein das Eingangssignal B auf 1 gesetzt wird. Werden beide Signale A und B zu 1 gesetzt, nimmt der Ausgang zunächst den erwarteten Wert Y=1 ein.
Wird der Eingang B wieder zurück zu 0 gesetzt, sollte der Ausgang Y den Wert 1 halten, er beginnt jedoch zu schwingen. Der Grund dafür ist, daß beim Übergang der Signale A,B,Y* von 111 zu 101 in der gezeigten Realisierung der Schaltung ein Hazard auftritt. Damit nimmt der Ausgang Y mit der Verzögerung der Gatter kurzzeitig fälschlicherweise den Wert 0 an. Da der Ausgang wieder auf den Eingang der Schaltung rückgekoppelt wird, tritt dieser Hazard nicht nur einmal auf, sondern führt zu dem beobachteten Schwingen der Schaltung. Die Schaltung schwingt dabei zwischen den Kombinationen A,B,Y*=101 und A,B,Y*=100.
Das hier beobachtete Verhalten hängt sehr stark von den Eigenschaften der Grundbauelemente (z.B. UND-, ODER-Gatter, Inverter) ab, mit denen die Schaltung

aufgebaut wird. Es kann durchaus sein, daß statt des Schwingens der Ausgang unerwartet den Wert 0 annimmt oder beim Test auch das erwartete Verhalten auftritt.
Insbesondere der letzte Fall ist sehr problematisch, da zunächst angenommen wird, die Schaltung sei fehlerfrei. Durch Temperaturänderung oder Alterung können sich jedoch die Eigenschaften der Bauteile ändern, so dass erst dann die Schwachstelle im Design zu einem fehlerhaften Verhalten der Schaltung führt.

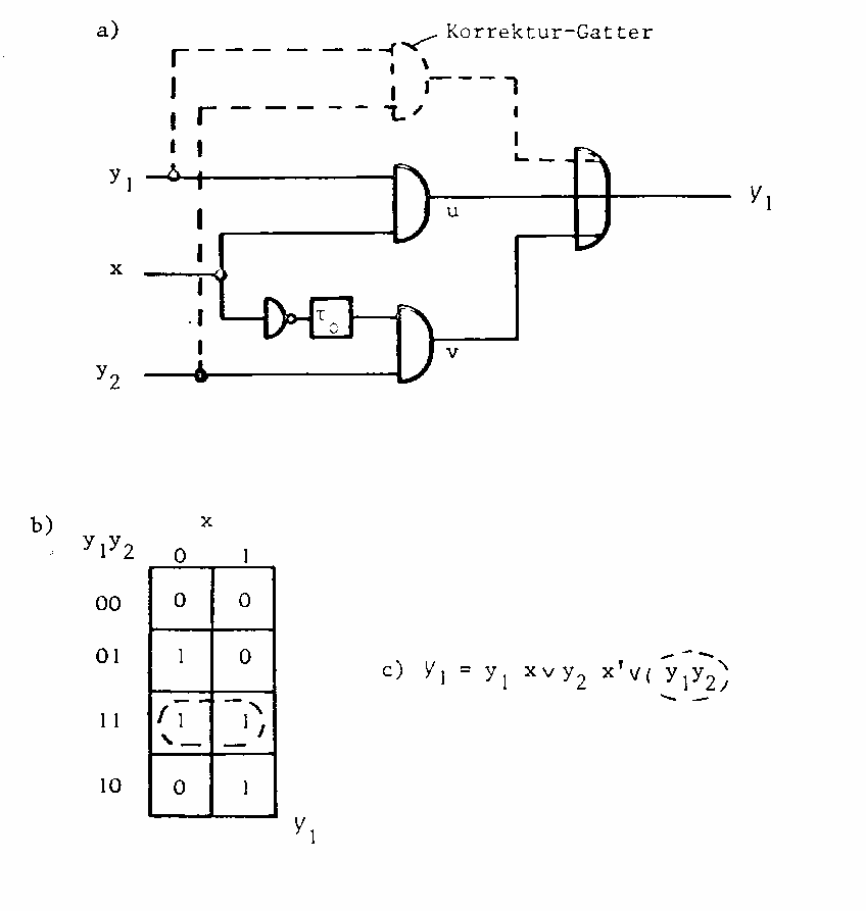
12.1.1.3 Eliminieren von kombinatorischen Hazards

Das Eliminieren von Hazards erfolgt durch Hinzufügen von Primimplikanten zur Minimalform, welche in den Übergangssituationen dafür sorgen, daß der Ausgang stabil bleibt.
Der Hazard tritt beim Übergang von einer Eingangskombination zu einer zweiten Kombination auf, wenn dabei ein Wechsel von einem Primimplikanten zu einem zweiten Primimplikanten erfolgt. Betrachten wir nochmals das vorhergehende Beispiel im KV-Diagramm:



Man erkennt deutlich, daß die Eingangskombination 001 zu PI_1 und 101 zu PI_3 gehört. Da im Falle der´Kombination 001 der Primimplikant PI_1 den Ausgangswert Y=1 erzeugt, im Falle 101 jedoch der Implikant PI_3, kommt es zu dem Problem.
Sorgt man nun jedoch dafür, daß ein zusätzlicher Primimplikant der Gleichung hinzugefügt wird, welcher sowohl bei der Kombination 001 als auch bei 101 den Ausgangswert Y=1 erzeugt, wird dieser Hazard nicht mehr auftreten. Im Beispiel kann PI_2 genau diese Aufgabe übernehmen. Somit ergibt sich die hazardfreie Gleichung der betrachteten Funktion zu:
$Y = PI1 \veebar PI2 \veebar PI3 = (X1 \wedge (\neg X3)) \veebar (X1 \wedge (\neg X2)) \veebar ((\neg X2) \wedge X3)$.

12.1.1.4 Weiteres Beispiel zum Eliminieren von kombinatorischen Hazards



12.1.1.5 Regeln zum Eliminieren von kombinatorischen Hazards

Zusammenfassend geht man beim Entwurf einer hazardfreien Beschreibung einer Funktion wie folgt vor:

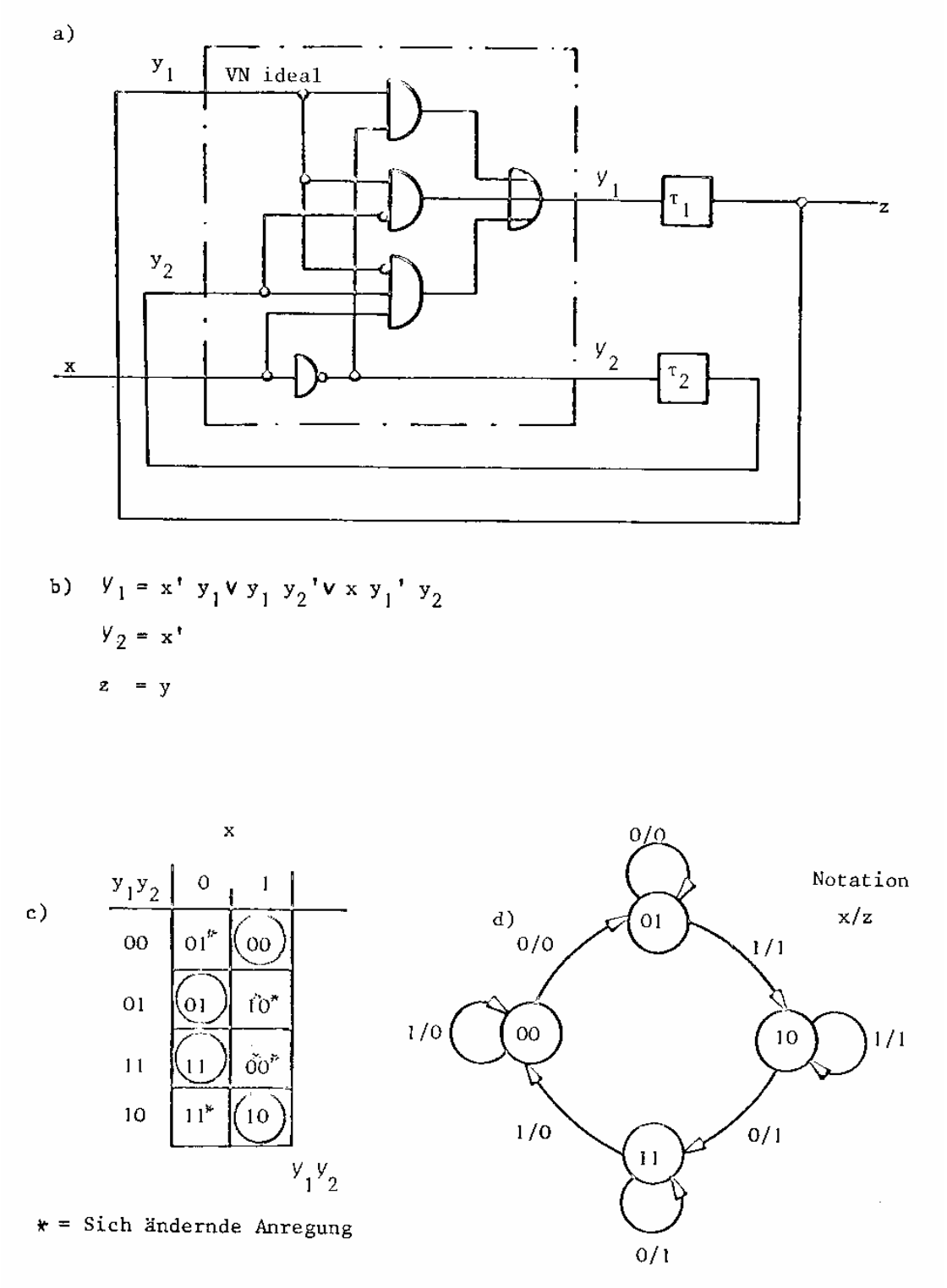
1. Erzeugen der Minimalform einer gegebenen booleschen Funktion.
2. Überprüfen der verwendeten Primimplikanten, ob durch einen Einkomponentenübergang (Änderung einer einzelnen Eingangsvariablen) ein Wechsel von einem aktiven Primimplikanten auf einen anderen Primimplikanten erfolgt. Solch ein Wechsel ist eine potentielle Gefahrenstelle zum Auftreten von Hazards.
3. Hinzufügen von redundanten Primimplikanten zur Minimalform, so daß die Hazard-gefährdeten Einkomponentenübergänge durch diese zusätzlichen Primimplikanten überdeckt und damit ungefährlich gemacht werden.
4. Zerlegen der Komponenteneingänge
5. Synchrone Schaltungen („takten“)

12.1.1.6 Funktions-Hazards:

Verursacht durch den gleichzeitigen Signalwechsel an mehreren Eingängen, Mehrkomponentenübergang
• Sind mit der Funktion der Schaltung verbunden
• Können bei jeder Schaltungsimplementierung auftreten
Bedingung für das Auftreten:
Es muß sich mehr als ein Eingang „gleichzeitig“ ändern

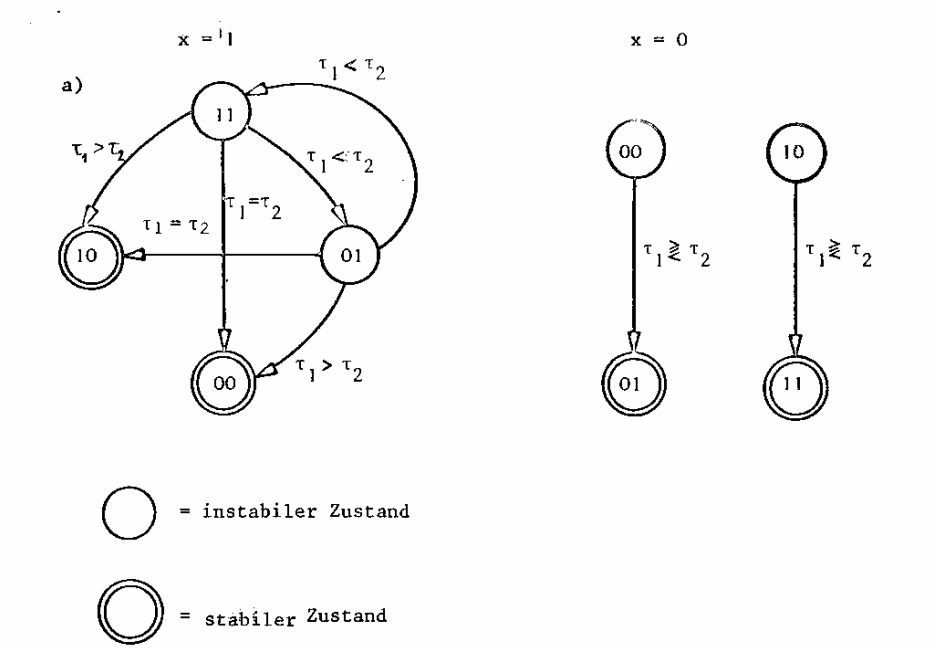
ideal: Alle Eingänge ändern sich gleichzeitig
real: Es finden Zeitverschiebungen statt, z.B. um 4 ns
-> Sequenzen von Einzeländerungen, die sukzessive stattfinden, 'Races'
Die Folge von Hazards können Fehlfunktionen von digitalen Systemen sein, z.B. falsche Adressierung von Datenspeichern

In der untenstehenden Schaltung gibt es zwei Rückführungen mit unterschiedlichen Verzögerungen. In der Tabelle erhält man vier stabile Zustände, die in der Folge (y1, y2) 00, 01, 10 11, 00 durchlaufen werden müssten. Man erkennt an zwei Stellen Mehrkomponenten-Übergänge. Aus dem stabilen Zustand 01 erzeugt x= 0->1 die Anregung (Y1,Y2) = 10. Sie weicht von der Anregung im stabilen Zustand Y1Y2 = 01 in beiden Komponenten ab. Welcher Zustand sich als nächstes einstellt, hängt vom Ausgang eines Wettlaufes (race) ab, d.h. von den Laufzeitdifferenzen in den Rückführungen. Je nachdem welches τ größer ist, können sich unterschiedliche Folgezustände einstellen.

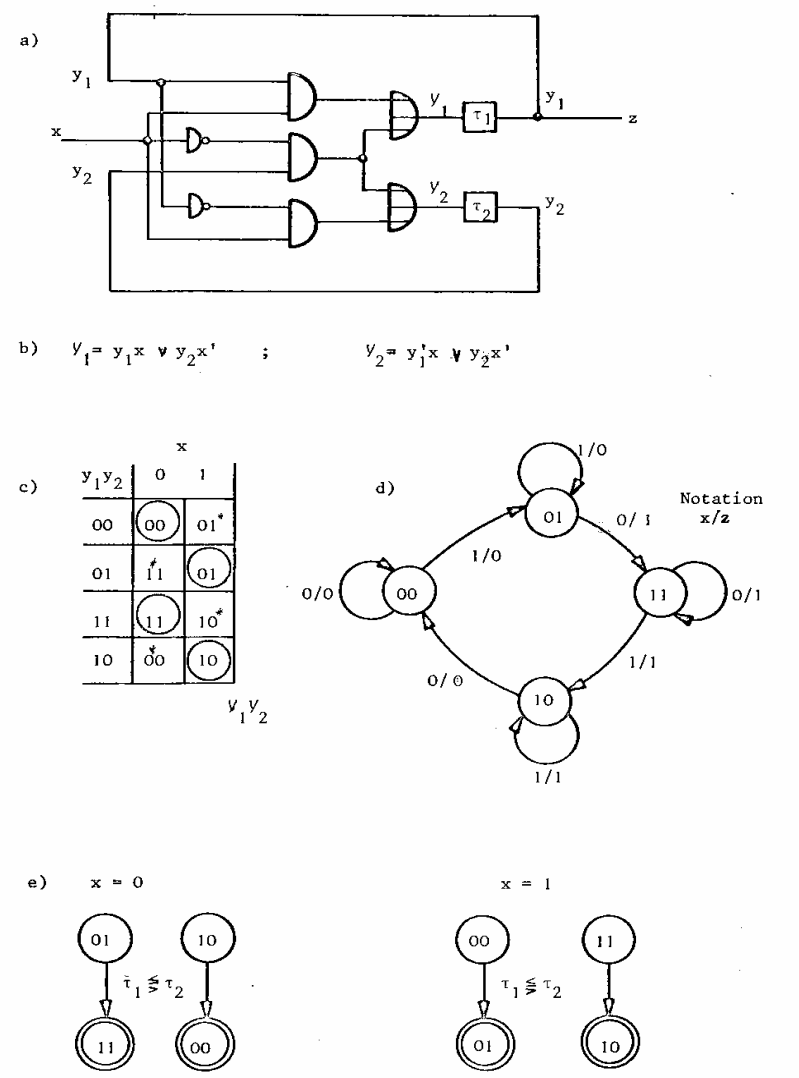


Binärzähler mit kritischen Races und Schwingzuständen
a) Schaltung, b) Zustandsgleichungen, c) Übergangstabelle, d) Zustandsdiagramm für $\tau_1=\tau_2$

Die kritischen Läufe lassen sich wie folgt darstellen:



Ein weiteres Beispiel zeigt einen Binärzähler ohne Läufe und Schwingzustände:



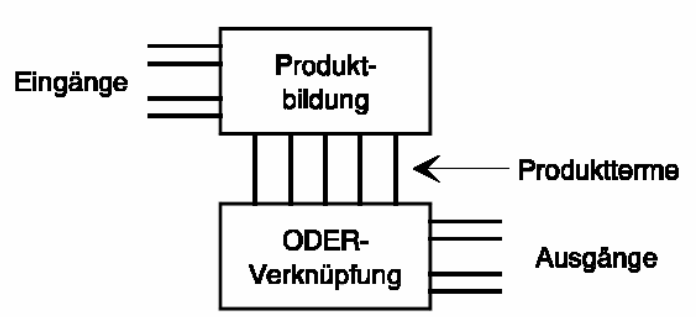
12.1.1.7 Regeln zum Eliminieren von funktionalen Hazards

Generell lassen sich Funktions-Hazards durch 2 Maßnahmen vermeiden:

- Verwandlung von Mehrkomponenten-Übergängen in mehrere Einkomponenten-Übergänge
- Einsatz synchroner Schaltungen

13 Programmierbare Bausteine

Wie wir wissen, läßt sich jede Schaltfunktion als distributive Normalform darstellen. Diese Form ist zweistufig. Zuerst werden Produkte gebildet (UND), dann werden die Produkte ge-Odert (von den Negationen einmal abgesehen). Man kann dies ausnutzen und flexible Bausteine für Systeme von Schaltfunktionen (bei vorgegebener Variablenzahl) zu realisieren. Dabei sind die Eingänge der UNDGatter (Produkte) und/oder der ODER-Gatter (Summen) nicht, wie in den bisherigen Beispielen, fest vorgegeben, sondern können nach Bedarf einzeln ausgewählt werden. Diese Bausteine sind, wie man sagt, programmierbar (PLD Progammable Logic Devices).



Programmierbare logische Schaltungen (PLD, Programmable Logic Devices) stellen eine logische Grundstruktur zur Verfügung, die vom Anwender durch Programmierung eine endgültige logische Funktion erhält. Zur Programmierung der Bausteine sind folgende Verfahren gebräuchlich:

13.1.1 PROM: Programmable Read Only Memory

Trennen einer Verbindung durch Durchbrennen einer Sicherung (Fuse). Der Baustein kann genau einmal programmiert werden, die Programmierung ist irreversibel. Enthält der Baustein Fehler, muß dieser gegen einen neuen, korrekten Baustein ausgetauscht werden.
Üblicherweise ist die UND-Matrix vollständig und fest vorgegeben, die ODER-Matrix ist programmierbar.
Ähnlich ist das Herstellen einer Verbindung durch Entfernen einer Isolierung (Antifuse). Auch dieser Programmiervorgang ist irreversibel.

13.1.2 EPROM: Erasable PROM,

Programmierung einer EPROM Zelle. Die Programmierung kann durch Bestrahlung des Bausteins mit UV-Licht wieder gelöscht werden. Daher besitzen Bausteine dieses Typs ein kleines Fenster, durch welches das Halbleiterblättchen sichtbar ist. Heute sind Bausteine dieses Typs kaum mehr gebräuchlich.

13.1.3 EEPROM: Electrical Erasable PROM

Programmierung einer EEPROM Zelle. Die Programmierung kann durch Anlegen elektrischer Signale wieder gelöscht werden. Dazu müssen moderne Bausteine nicht von der Platine entfernt werden, sondern sowohl das Löschen als auch das Programmieren ist in der Schaltung (In-Circuit Programming) möglich.

Eingebettete, flüchtige Speicherzellen. Die Information zum Einstellen der Logik wird bei Bausteinen dieses Typs in flüchtigen Speicherzellen abgelegt. Diese schalten die zugehörige Hardware in die gewünschte Funktion. Beim Ausschalten der Versorgungsspannung geht die Information verloren, daher müssen diese Bausteine bei jedem Booten des Systems neu programmiert werden.

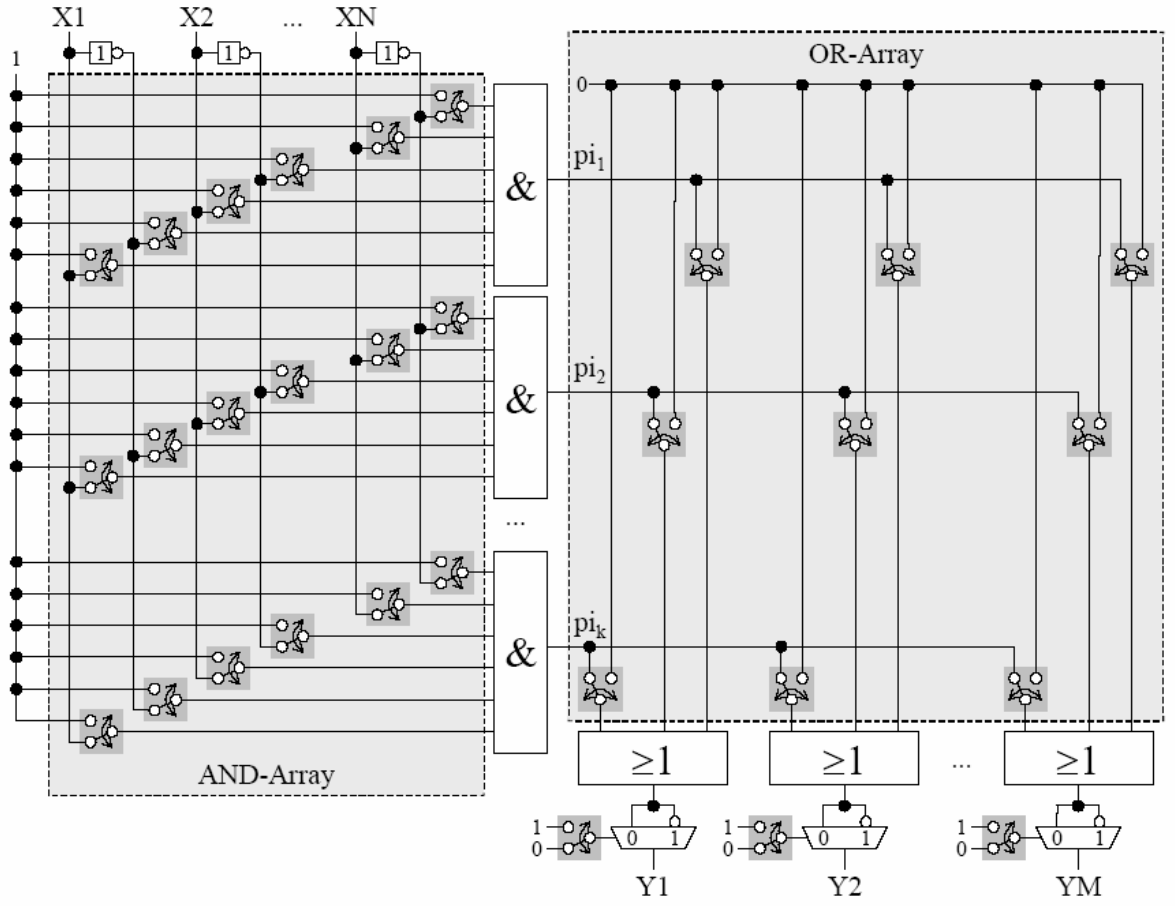
Die programmierbare Grundstruktur der Bausteine unterscheidet sich ebenfalls grundsätzlich. Es gibt zwei wichtige Klassen. Diese sind Bausteine mit UND/ODER-Struktur. Zu diesen Bausteinen gehören die klassischen, einfachen

13.1.4 PALund GAL-Bausteine

sowie moderne, hochkomplexe und schnelle CPLD-Bausteine.
Bausteine mit Logikzellen. Diese Bausteine werden auch als FPGAs bezeichnet

13.1.5 Programmierbare Bausteine mit UND/ODER-Struktur

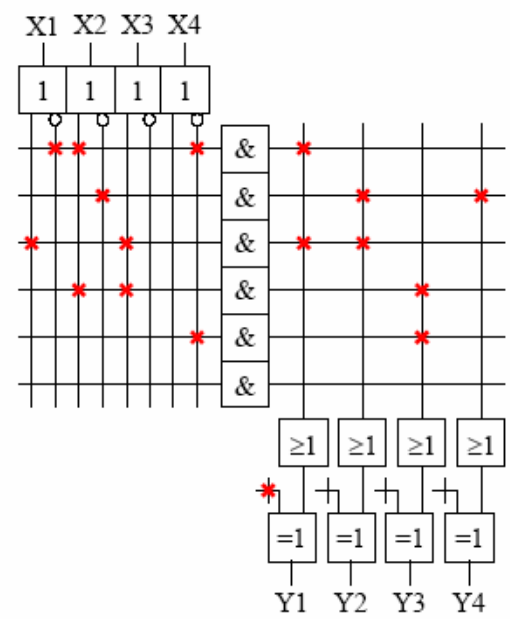
Die Grundschaltung programmierbarer Bausteine mit UND/ODER-Struktur besteht aus einem programmierbaren AND-Array, über welches die Eingänge an eine erste Gatterstufe aus UND-Gattern angeschlossen wird. Es folgt ein programmierbares OR-Array, welches die Ausgänge der UND-Gatter mit einer zweiten Stufe aus ODER-Gattern verbindet. Am Ausgang der ODER-Gatter sind programmierbare Ausgangsinverter vorgesehen:



Über das AND-Array können die Eingänge über Schalter in negierter oder nicht-negierter Form an die UNDGatter angeschlossen werden. Nicht benutzte Eingänge werden deaktiviert, z.B. durch Verbinden mit dem Signalwert 1. Mit jedem UND-Gatter kann somit ein beliebiger Implikant oder Minterm realisiert werden.
Über das OR-Array werden die Implikanten den ODER-Gattern zugeführt und dort zur Gesamtfunktion verknüpft. Auch im ODER-Feld werden nicht benutzte Einngänge der ODER-Gatter deaktiviert, z.B. durch Verbinden mit dem Signalwert 0.

13.1.6 PLA: Programmable Logical Array

Mit dem frei programmierbaren AND- und OR-Array können Funktionen in disjuktiver Normalform realisiert werden. Durch die programmierbaren Ausgangsinverter ist aufgrund des Shannonschen Gesetzes auch die Realisierung der konjunktiven Form möglich. Üblicherweise wird diese logische Grundstruktur in einer kompakten Form dargestellt, wo bei jedem Gatter eine Linie die Gattereingänge symbolisiert. Eine Markierung auf einem Eingangssignal und der zugehörigen Gatterlinie zeigt, daß das Eingangssignal auf das Gatter geführt wird. Nachfolgendes Beispiel zeigt die kompakte Form und demonstriert die Realisierung einiger logischer Funktionen:

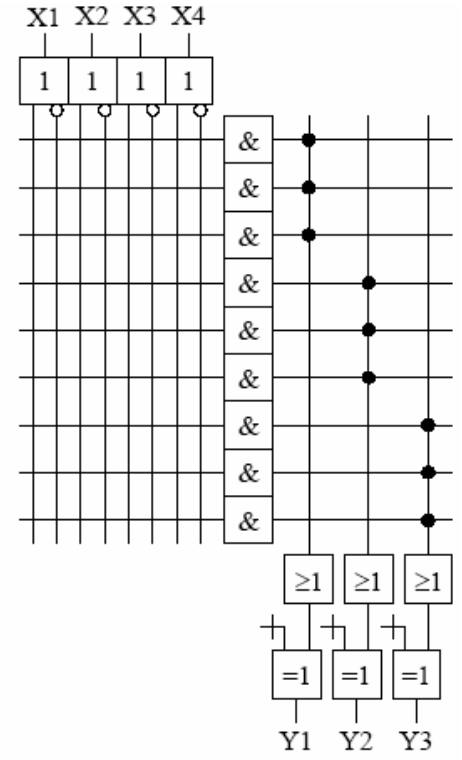


$$Y1 = \overline{(\overline{X1} \wedge X2 \wedge \overline{X4}) \vee (X1 \wedge X3)}$$
$$= (X1 \vee \overline{X2} \vee X4) \wedge (\overline{X1} \vee \overline{X3})$$
$$Y2 = \overline{X2} \vee (X1 \wedge X3)$$
$$Y3 = (X2 \wedge X3) \vee \overline{X4}$$
$$Y4 = \overline{X2}$$

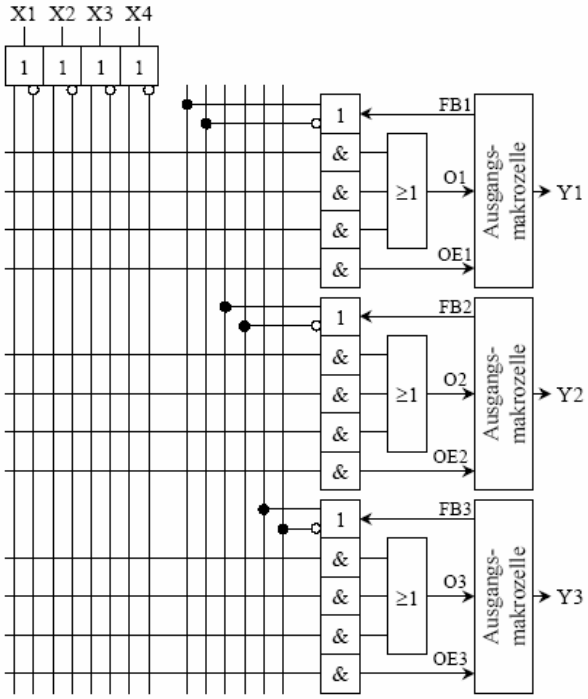
Die vorliegende Form, in der sowohl das AND- als auch das OR-Array programmiert werden kann, findet sich in PLA-Bausteinen wieder. Dort können Implikanten in mehreren Funktionen gemeinsam genutzt werden. Die Verwendung von PLA-Bausteinen ist jedoch eher selten. Die gebräuchlichen PAL- und Speicher-Bausteine schränken die Möglichkeiten der Programmierung auf ein sinnvolles Maß ein, so daß deren Verwendung gut handhabbar ist.

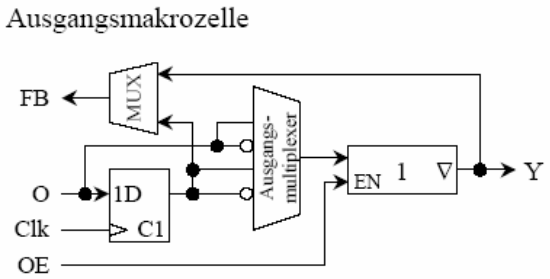
13.1.7 PAL-Bausteine (Programmable Array Logic)

Bei PAL-Bausteinen (dazu gehören auch GAL-Bausteine Generic Array Logic, die PAL-Struktur besitzen aber vom Hersteller mit der eigenen Bezeichnung GAL versehen wurden) ist das OR-Array fest programmiert, so daß die UND-Gatter der ersten Stufe fest den ODER-Gattern am Ausgang zugeordnet werden.
Nachfolgende Abbildung zeigt eine kleine PAL-Struktur mit 4 Eingängen und 3 Ausgängen, in der jeweils 3 UND-Gatter einem Ausgang zugeordnet werden.



Einfache PAL-Bausteine besitzen mindestens 8 Eingänge und Ausgänge.
Bei modernen PAL-Bausteinen findet man an den Ausgängen komplexe, programmierbare Makrozellen die neben der Invertierung des Ausgangssignals eine Speicherung in einem Flip-Flop ermöglichen und auch die Rückführung des Ausgangssignals in das programmierbare AND-Array ermöglichen.
Die nachfolgende Abbildung zeigt die erweiterte Struktur:



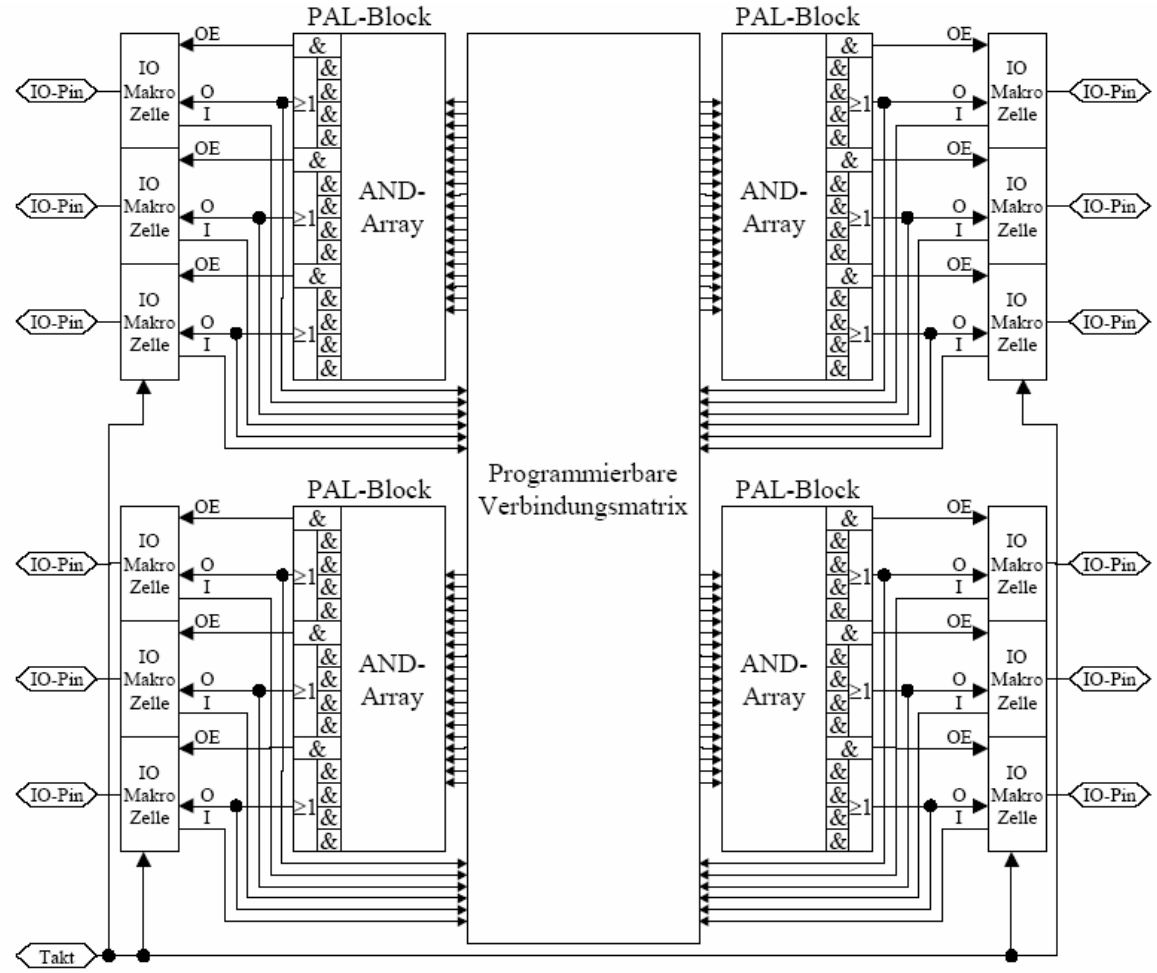


Statt des OR-Arrays sind fest verschaltete ODER-Gatter gezeigt. Der Ausgang O eines ODER-Gatters wird in die Ausgangsmakrozelle geführt. Ebenso ein zusätzlicher Produktterm OE.

Aus der Ausgangsmakrozelle führt ein Rückkopplungssignal FB (FB: Feed Back), welches rückwärts wieder in das AND-Array geführt ist und dort in die Produktterme eingebunden werden kann.
Das Bild der Ausgangsmakrozelle zeigt, daß das Ausgangssignal O des ODER-Gatters entweder direkt ausgegeben oder in einem flankengesteuerten D-Flip-Flop zwischengespeichert werden kann. Auf den Ausgang Y wird entweder das Signal O oder der Ausgang des Flip-Flops geschaltet, dabei kann jeweils zwischen dem negierten und nicht-negierten Signal ausgewählt werden. Die Auswahl erfolgt mit dem programmierbaren Ausgangsmultiplexer.
Hinter dem Ausgangsmultiplexer ist ein Tri-State Bustreiber angeordnet, der mit dem programmierbaren Produktterm OE geschaltet wird.
Das Rückkopplungssignal FB wird mit einem zweiten, programmierbaren Multiplexer erzeugt, dieser wählt entweder den Ausgang Y oder den Flip-Flop-Ausgang als Rückkopplungssignal aus.
Es ist zu bemerken, daß bei konstant deaktiviertem Tri-State Treiber am Ausgang (OE=0) und Auswahl des Ausgangs Y als Rückkopplungssignal FB der Ausgang Y wie ein normaler Eingang verwendet wird.

13.1.8 CPLD-Bausteine

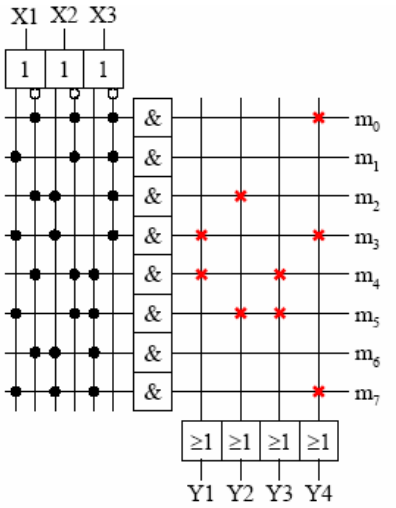
CPLD-Bausteine gruppieren mehrere einfache PAL-Blöcke um eine programmierbare Verbindungsmatrix herum. Zur Ein- und Ausgabe von Signalen sind, ähnlich wie bei den komplexen PAL-Bausteinen, Makrozellen vorhanden.
Nachfolgende Abbildung zeigt die Architektur der CPLD-Bausteine:



Über die Eingangssignale I gelangen externe Signale an die programmierbare Verbindungsmatrix. In der Verbindungsmatrix können beliebige Verbindungen zwischen Ein- und Ausgängen geschaltet werden. Die Ausgänge der Verbindungsmatrix bilden die Eingänge der PAL-Blöcke. Typische PAL-Blöcke besitzen ca. 50 Eingänge und 10-20 Ausgänge, wobei jeder Ausgang aus bis zu 10-15 Produkttermen gebildet wird.
Anders als in komplexen PAL-Bausteinen werden die Ausgänge der UND/ODER-Struktur direkt und nicht über die Makrozelle zurückgekoppelt. Die Rückkopplung erfolgt auch nicht auf das eigene AND-Array, sondern auf die programmierbare Verbindungsmatrix. Von dort aus kann das Signal natürlich wieder in das eigene ANDArray,
aber auch in andere PAL-Blöcke des Bausteins eingespeist werden. An den Ausgängen der PAL-Blöcke können programmierbar Flip-Flops zugeschaltet werden, diese sind in obigem Blockschaltbild nicht gezeigt.
Damit ist ein synchrones Rückführen der PAL-Ausgänge möglich.
Die IO-Makrozellen besitzen eine ähnliche Struktur wie Makrozellen der komplexen PAL-Bausteine. Das bei der Beschreibung der PAL-Bausteine als Rückkopplungssignal FB bezeichnete Signal ist hier als Eingangssignal I bezeichnet. Ebenso wie bei den PAL-Bausteinen kann über einen Produktterm ein Tri-State Treiber am Bausteinausgang geschaltet werden.
Es muß darauf hingewiesen werden, daß die gezeigte Struktur lediglich die Grundstruktur von CPLDBausteinen widerspiegelt. Reale Bausteine weisen eine Fülle von Besonderheiten sowie zusätzlichen Verbindungen und Signalen auf. Zur detaillierten Information wird daher auf die Datenblätter der Anbieter von CPLD-Bausteinen verwiesen. Anbieter von CPLD-Bausteinen sind beispielsweise die Firmen Altera (www.altera.com), Cypress (www.cypress.com), XILINX (www.xilinx.com), Lattice (www.latticesemi.com).

14 Speicherbausteine als programmierbare Logik

Bei Speicherbausteinen ist das AND-Array fest programmiert. Diese Architektur besitzt bei n Eingängen genau
N=2n UND-Gatter.
Mit der festen Programmierung des AND-Arrays wird mit dem ersten UND-Gatter der Minterm m0 erzeugt, mit dem nächsten Gatter der Minterm m1 bis schließlich das letzte UND-Gatter den Minterm mN-1 erzeugt. Da alle Minterme zur Verfügung stehen kann durch Programmierung des OR-Arrays mit jedem ODER-Gatter eine beliebige boolsche Funktion erzeugt werden.
Nachfolgende Abbildung zeigt einen Speicher mit 3 Ein- und 4 Ausgängen. Es besteht intern aus 23 UNDGattern, mit denen die Minterme m0 bis m7 erzeugt werden.



$$Y1 = m_3 \vee m_4 = (X1 \wedge X2 \wedge \overline{X3}) \vee (\overline{X1} \wedge \overline{X2} \wedge X3)$$
$$Y2 = m_2 \vee m_5 = (\overline{X1} \wedge X2 \wedge \overline{X3}) \vee (X1 \wedge \overline{X2} \wedge X3)$$
$$Y3 = m_4 \vee m_5 = (\overline{X1} \wedge \overline{X2} \wedge X3) \vee (X1 \wedge \overline{X2} \wedge X3)$$
$$Y4 = m_0 \vee m_3 \vee m_7$$
$$= (\overline{X1} \wedge \overline{X2} \wedge \overline{X3}) \vee (X1 \wedge X2 \wedge \overline{X3})$$
$$\vee (X1 \wedge X2 \wedge X3)$$

Beispielhaft ist die Programmierung einiger Funktionen gezeigt.
Es bleibt anzumerken, daß der Aufbau asynchroner Schaltungen mit Speichern nicht zu empfehlen ist. Da alle Funktionen aus einzelnen Mintermen aufgebaut werden, können beim Übergang von einem Minterm zu einem zweiten Minterm immer Hazards auftreten, die das Verhalten einer asynchronen Schaltung unvorhersehbar machen.
Als Speicherbausteine werden Bausteine mit nicht-flüchtigem und auch mit flüchtigem OR-Array eingesetzt. Erste werden als PROM bezeichnet. PROM-Bausteine werden auch in einem anderen Kontext eingesetzt. Sie dienen als Festwertspeicher für Rechner, deren Inhalt auch beim Ausschalten der Versorgungsspannung erhalten bleibt. Häufig werden LUT mit Speicherbausteinen implementiert, unter jeder Adresse (Eingangswert) kann ein beliebiger Wert nachgesehen (look up) werden. Gut zum freien Umkodieren von Daten.
Auf Speicher wird nochmals in einem separaten Abschnitt eingegangen.

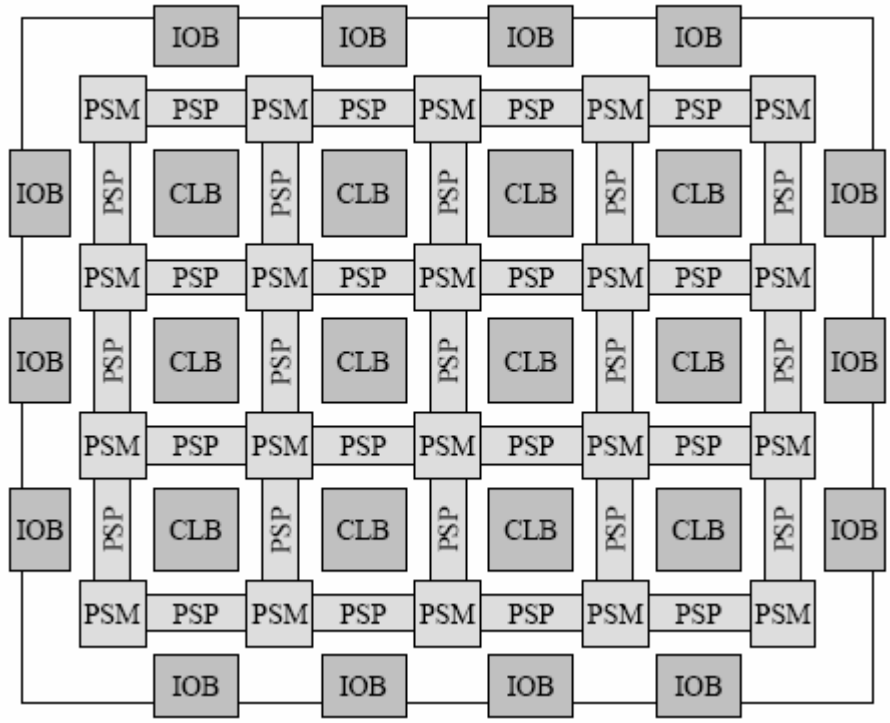
14.1.1 Zusammenfassung

Analysiert man die unterschiedlichen Bausteintypen mit UND/ODER-Struktur so ergibt sich als großer Vorteil, daß durch die regelmäßige Struktur und die festen Verbindungspfade die Verzögerungszeit td von Schaltungen gut vorhergesagt werden kann. Weiterhin können die bekannten Entwurfsverfahren für logische Funktionen direkt auf die eingebetteten UND/ODER-Schaltungen angewendet werden. Nachteil der Schaltungen ist die global angelegte Struktur der AND- und OR-Arrays bei PALs und PROMS und des programmierbaren Verbindungsnetzwerks der CPLD-Bausteine. Werden nur lokale Strukturen benötigt, wird die Performance dieser Hardwareressource nur schlecht ausgenutzt. Weiterhin ist die Realisierung solcher zentraler, globaler Ressourcen technologisch aufwendig und limitiert die erreichbare Komplexität der Bausteine.

15 FPGAs: Field Programmable Gate Array

Das Grundelement von FPGA-Bausteinen sind kleine, programmierbare Logikzellen (CLBs Configurable Logic Block), die in ihrer Funktion programmiert und mittels kleiner Verbindungselemente programmierbar zusammengeschaltet werden können. Die Herstellung von Verbindungen erfolgt über Routing-Kanäle, die aus programmierbaren Verbindungspunkten PSP (Programmable Switching Points) und kleinen Schaltmatrizen PSM (Programmable Switching Matrix) aufgebaut sind. Die Ein- und Ausgabe erfolgt mittels IO-

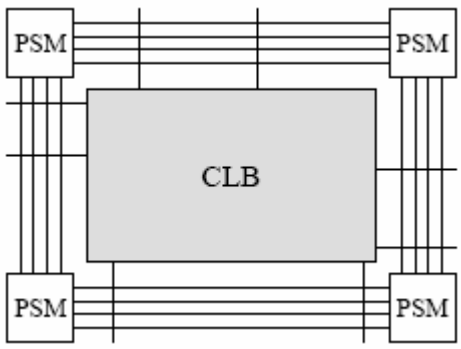
Blöcken (IOB), die ebenfalls an die Verbindungselemente angeschlossen sind. Die Zellen sind im FPGA in einer Matrix angeordnet. Damit erhält man die folgende Übersichtsstruktur.



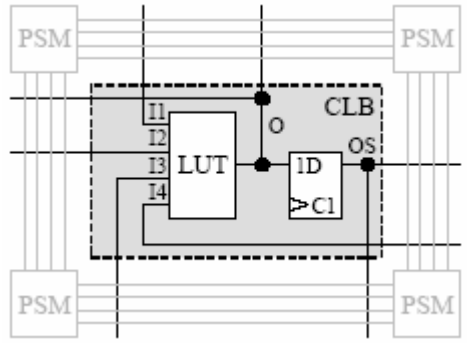
Im Inneren des FPGA-Bausteins liegen die programmierbaren Logikblöcke CLB, die über die PSP Anschlusspunkte programmierbar an die Routing-Ressourcen angeschlossen werden. Das Routing der Verbindungen erfolgt durch Programmierung der PSM Matrix. Die IOB Zellen sind am Rand des Chips angeordnet und können ebenfalls programmierbar an die PSP Punkte angeschlossen werden.

15.1.1 Ein einfaches Modell-FPGA

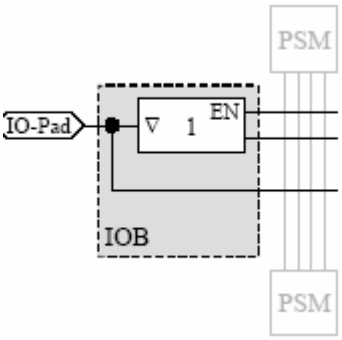
Zur detaillierten Erläuterung der FPGA-Funktionalität wird ein einfaches Modell eines FPGAs verwendet, welches die wichtigesten Eigenschaften eines FPGAs aufzeigt.
Das Modell-FPGA soll 4 Routing-Leitungen zwischen benachbarten PSM-Elementen besitzen. Eine Routing-Leitung führt von einem PSM zu einem benachbarten PSM, stellt also eine lokale Verbindung dar:



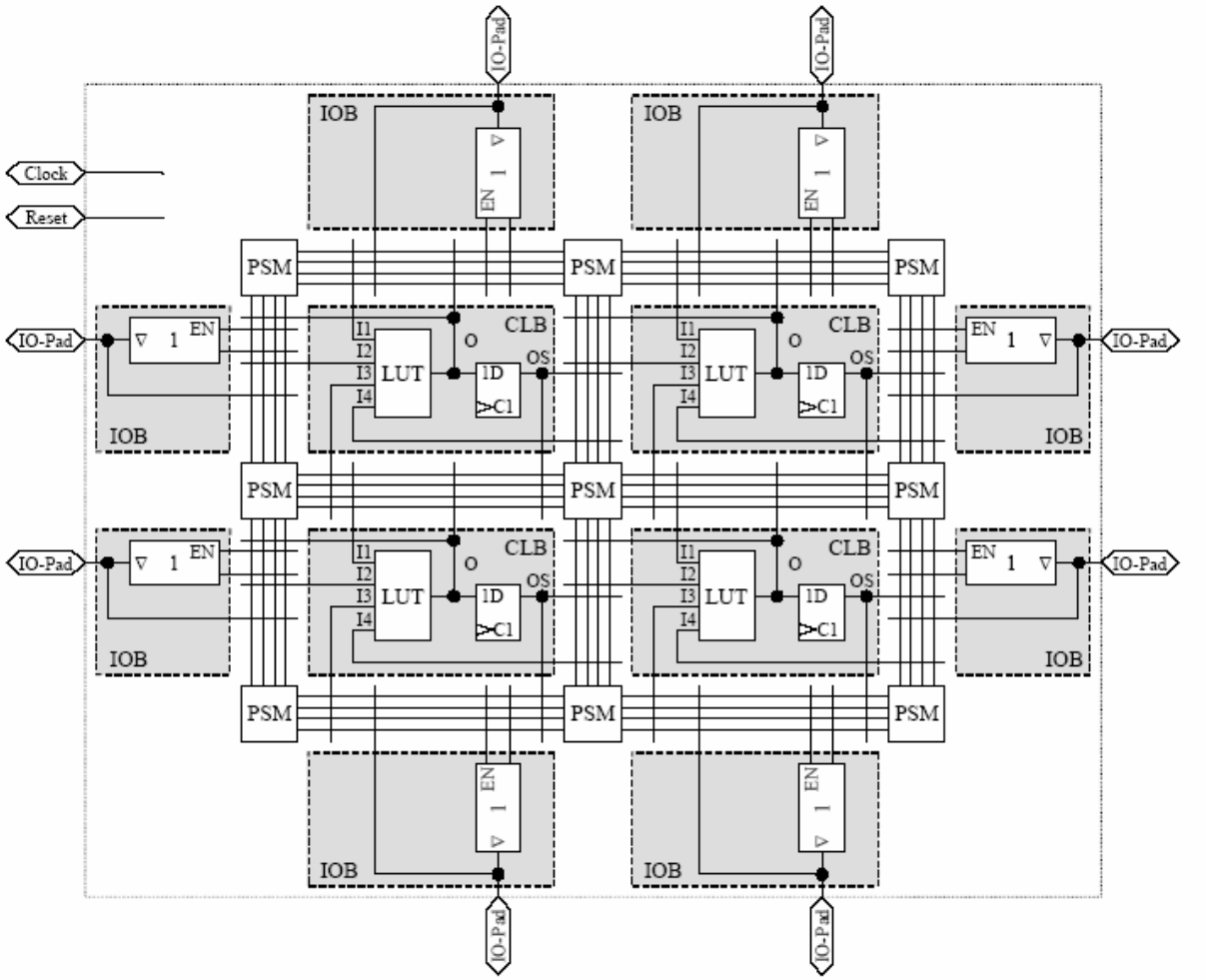
Reale FPGAs besitzen neben diesen lokalen Verbindungen auch globale Verbindungen, um Verbindungen zwischen weit entfernten Logikelementen im Baustein zu erleichtern. Im Modell-FPGA wollen wir darauf verzichten.
Man erkennt in der Abbildung, daß ein CLB Anschlussleitungen besitzt, die zu den benachbarten Routing-Leitungen führen. Die Kreuzungspunkte stellen, ähnlich wie bei den PLD-Bausteinen, programmierbare Verbindungspunkte dar. Durch Programmierung kann somit eine Anschlussleitung des CLBs mit einer oder mehreren Routing-Leitungen verbunden werden.
Für das Modell-FPGA soll der CLB die folgende Innenschaltung erhalten:



Von jeder der vier Seiten des CLBs wird eines der Eingangssignale I1, I2, I3 und I4 auf eine „Look-Up“-Tabelle LUT geführt.
Eine Look-Up-Tabelle ist ein kleiner Speicher, mit dem programmierbar die gewünschte Funktion
O = f(I1,I2,I3,I4) eingestellt wird. Im Modell-FPGA kann das direkte Ausgangssignal O oben und links an die Routing-Verbindungen angeschlossen werden. Das mit einem Takt über ein D-Flip-Flop synchronisierte Signal OS kann rechts und unten mit den Routing-Leitungen verbunden werden. Die Ausgangszelle des Modell-FPGAs wird sehr einfach gestaltet. Der Signalwert des Pins kann ausgelesen und ein Signal über einen Tri-State Treiber ausgegeben werden:

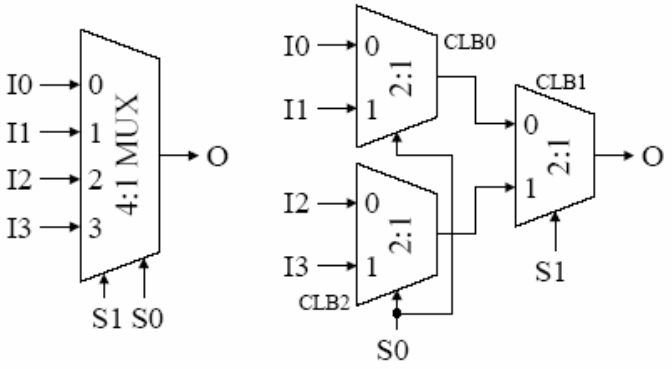


Mit diesen Grundelementen ergibt sich ein einfaches 4x4 Modell-FPGA:

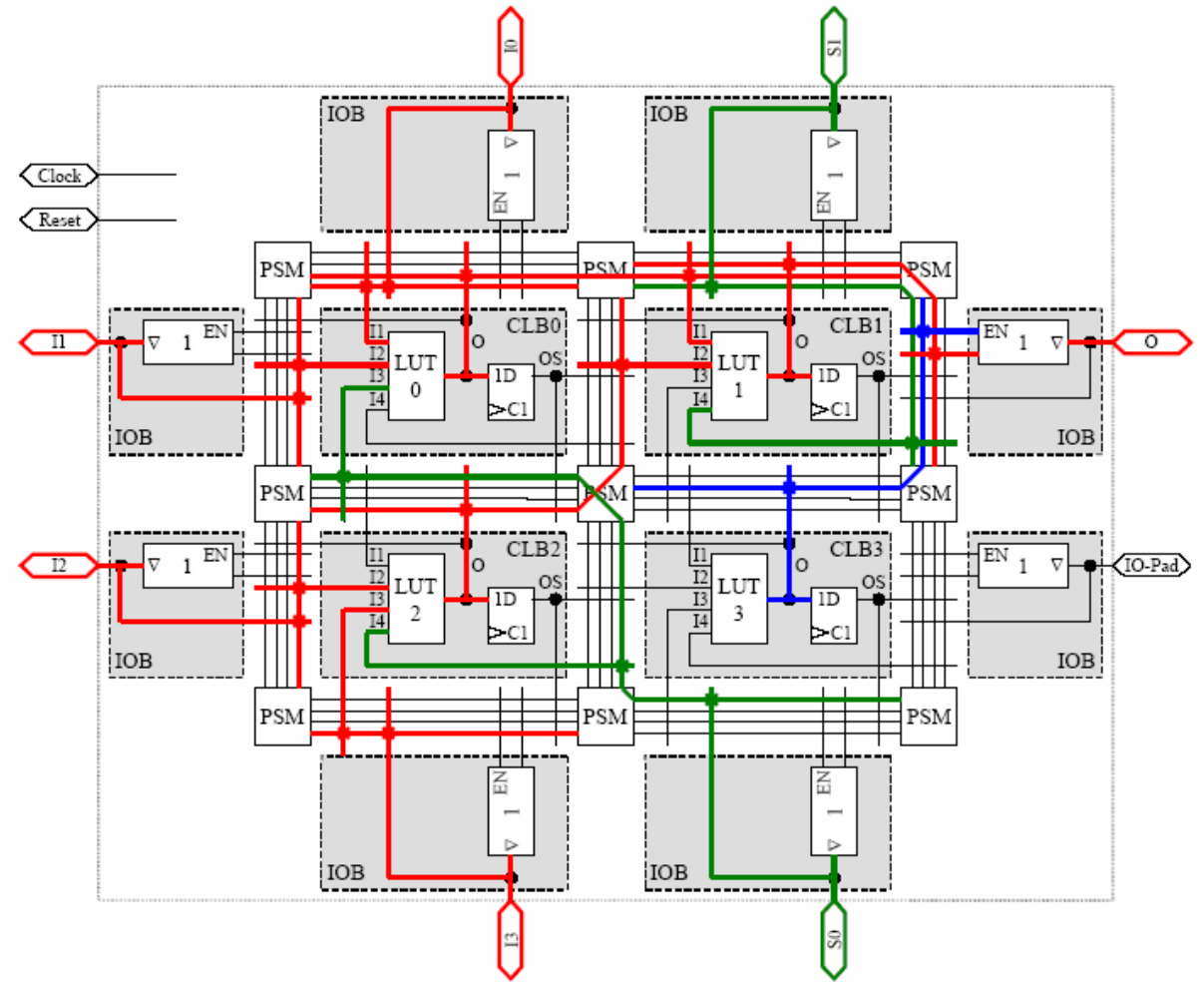


Die Signale Clock und Reset sind an jeden CLB geführt. Mit dem Signal Reset können alle Flip-Flops gemeinsam zurück auf 0 gesetzt werden. Das Signal Clock ist das gemeinsame Taktsignal für alle CLBFlip-Flops.
Zur Einstellung einer gewünschten logischen Funktion im FPGA müssen 3 verschiedene Ressourcen programmiert werden:
1. Einstellen der LUT-Funktionen in den CLBs.
2. Anschluss der CLB-Anschlüsse an die Routing-Leitungen mittels PSP-Punkten.
3. Verbinden mehrerer Routing-Leitungen (Segmente) zu einem gemeinsamen Signal mittels der PSM-Matrizen.
Im Gegensatz zur UND/ODER-Struktur ergeben sich durch diese verschiedenartigen Programmiermöglichkeiten eine große Vielzahl möglicher Lösungen für ein Problem. Typischerweise wird man keine optimalen oder minimalen Lösungen suchen, sondern eine Lösung ermitteln und dann nachprüfen, ob die erzielten Zeiteigenschaften zur Lösung der Aufgabe ausreichen.

Beispiel: Realisierung eines 4:1 Multiplexers mit dem Modell-FPGA
Die Programmierung eines FPGAs soll an einem einfachen Beispiel erläutert werden. Ein 4:1 Multiplexer soll mit dem gezeigten Modell-FPGA realisiert werden.
Da im verwendeten FPGA keine Logikelemente existieren, die 6 Eingangssignale verknüpfen können, muß die Funktion in Teilfunktionen zerlegt werden:



Mit der gezeigten Zerlegung in drei 2:1 Multiplexer kann jeder dieser kleinen Multiplexer auf einen CLB abgebildet werden. Eine willkürliche Zuordnung ist in der Abbildung eingetragen. Der vierte CLB wird verwendet, um auf den Freigabeeingang des Tri-State Treibers einen festen Logikpegel zu schalten, so daß der Treiber immer aktiv ist.
Die Schaltung kann nun auf den FPGA-Baustein abgebildet werden:



Die gezeigte Realisierung stellt eine unter vielen möglichen Realisierungen dar. Der Signalpfad von den Eingängen I0, I1, I2 und I3 zum Ausgang O ist rot dargestellt. Der Anschluss der Steuersignale S0 und S1 ist grün dargestellt. Die Erzeugung eines festen Pegels für den Ausgangstreiber ist mit blauen Signalen eingezeichnet.

Der Abbildung kann entnommen werden, wie die Programmierung der PSP- und PSM-Elemente erfolgt. Jede PSP-Verbindung ist mit einem kleinen Kreuz markiert, die PSM-Verbindungen sind auf die zugehörigen Elemente aufgezeichnet.
Für die CLB-Programmierung müssen die LUT-Funktionen noch spezifiziert werden

LUT0

I1 I2 I3 I4	O
0 - 0 -	0
1 - 0 -	1
- 0 1 -	0
- 1 1 -	1

LUT1

I1 I2 I3 I4	O
0 - - 0	0
1 - - 0	1
- 0 - 1	0
- 1 - 1	1

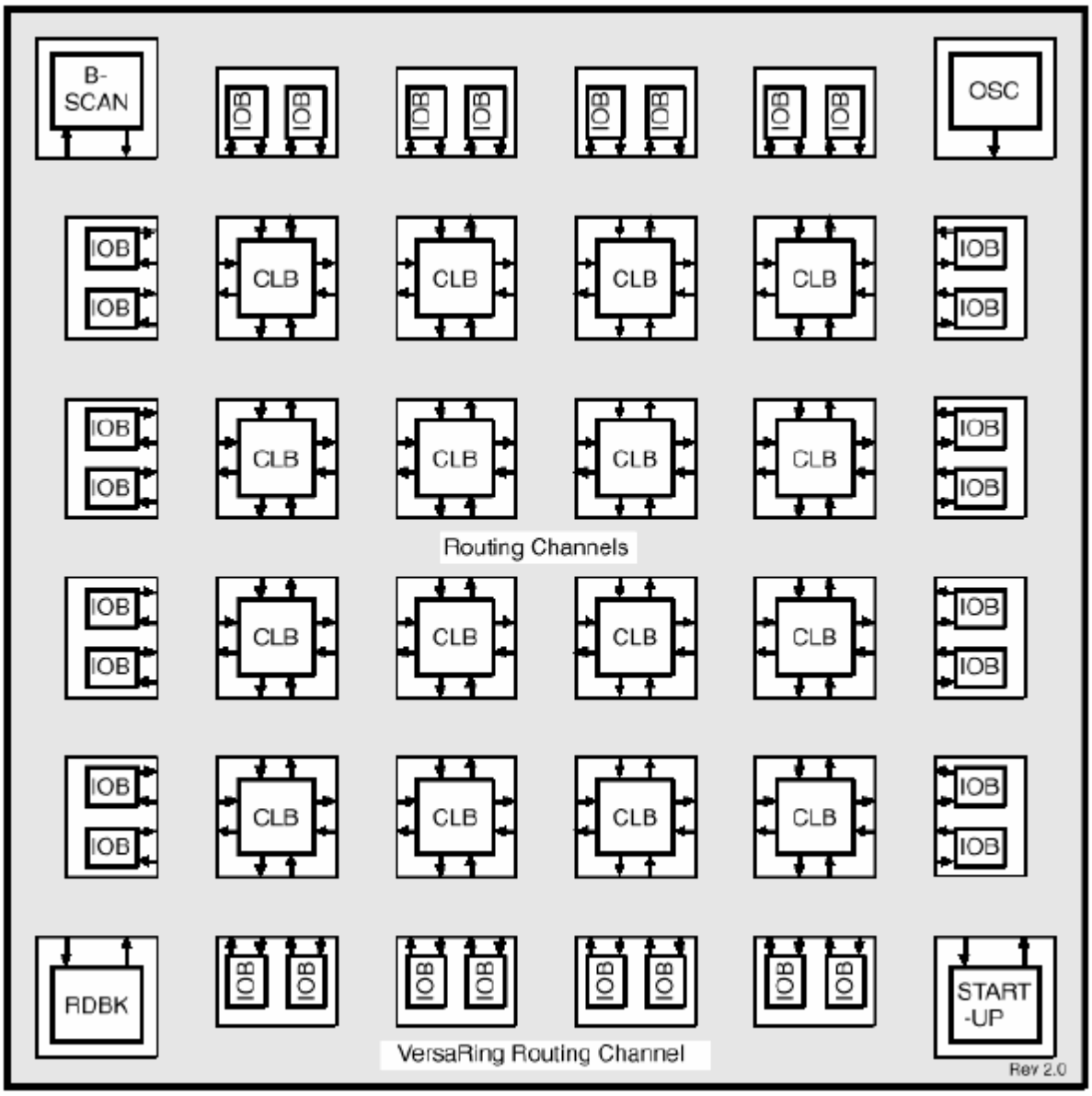
LUT2

I1 I2 I3 I4	O
- 0 - 0	0
- 1 - 0	1
- - 0 1	0
- - 1 1	1

LUT3

I1 I2 I3 I4	O
- - - -	1

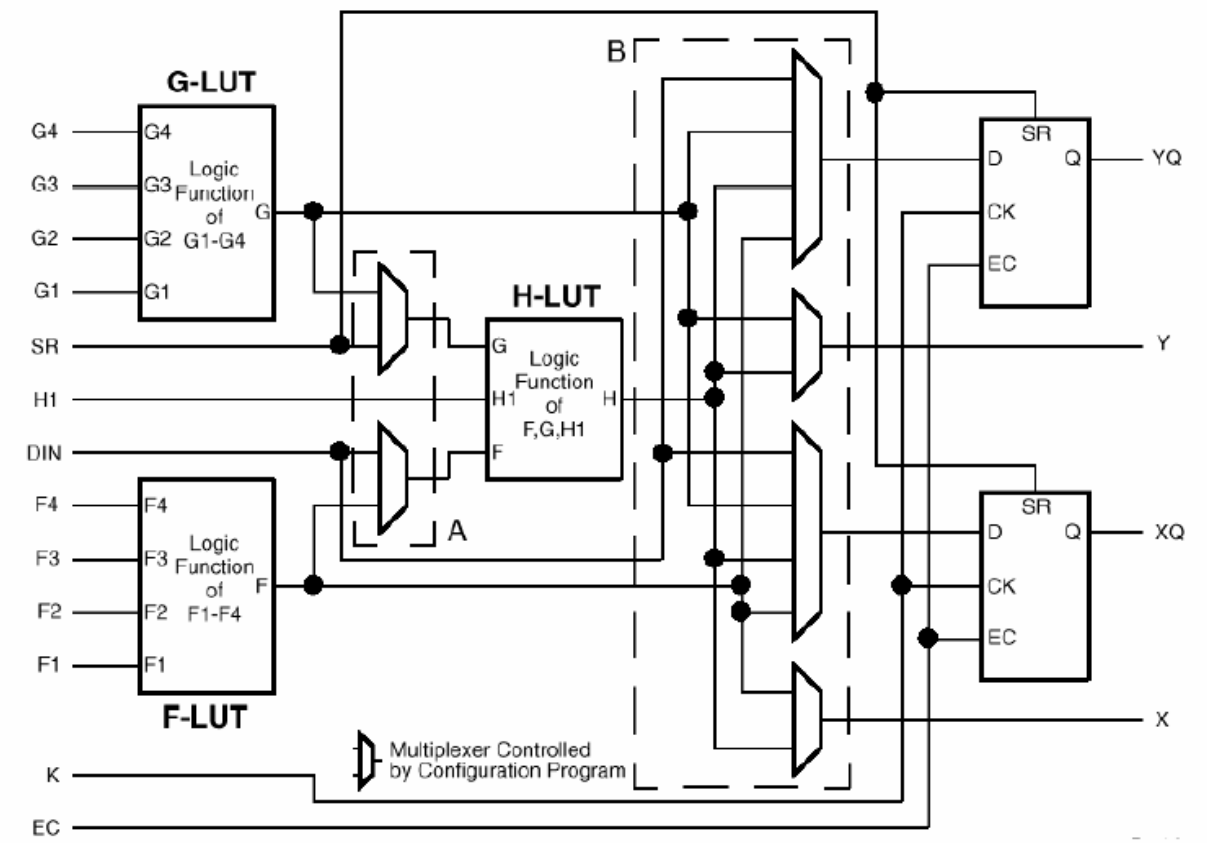
Ein Beispiel kommerzieller FPGAs: Die Spartan-Familie der Firma Xilinx
Der größte Anbieter von FPGA-Bausteinen ist die Firma Xilinx (www.xilinx.com). Sie bietet mit der Spartan-Familie hochkomplexe, preiswerte FPGAs an.
Die Übersicht zeigt die Grundstruktur aller Bausteine der Familie aus IOBs und CLBs sowie in den Ecken einige Sonderfunktionen zum Initialisieren, Testen und zur Taktaufbereitung, auf die nicht näher eingegangen werden soll. Die Bereiche zwischen den Logikblöcken sind als Routing Channels bezeichnet. Dahinter verbergen sich die PSP- und PSM-Elemente.



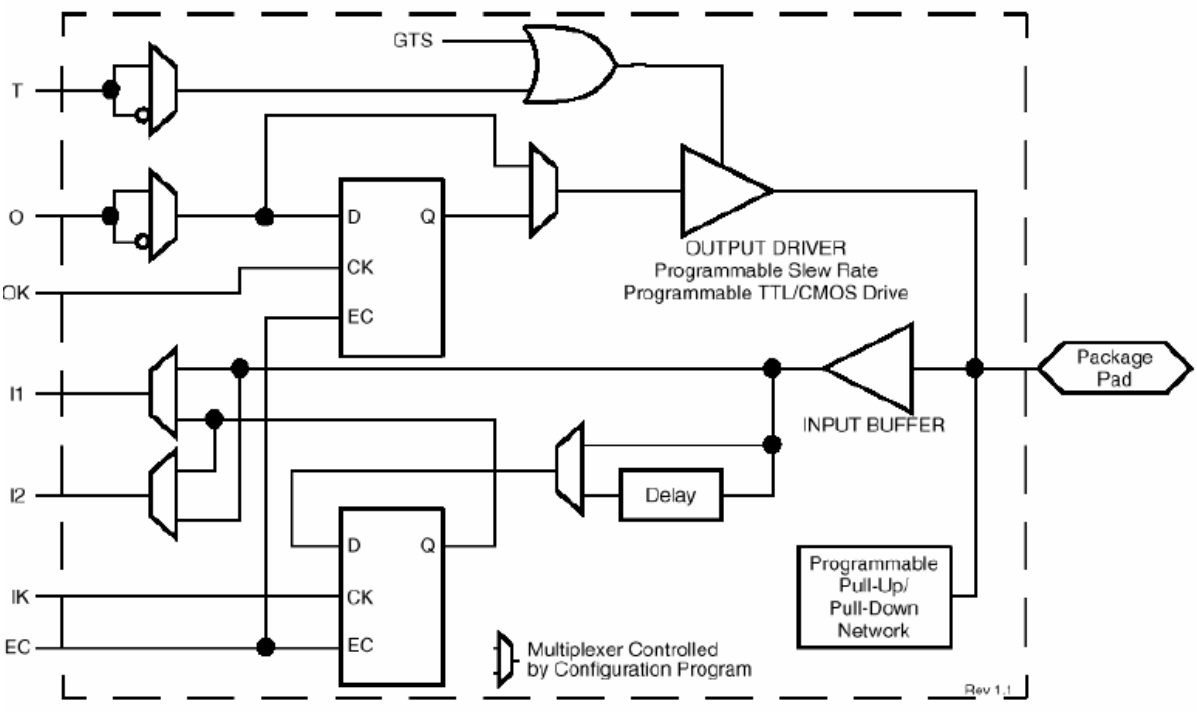
Die Anzahl der CLBs in einem Spartan-FPGA variiert. Nachfolgende Tabelle zeigt die Größen der Bausteine:

Bauteil	CLB Matrix	CLB-Anzahl	FlipFlop-Anzahl	Max. User I/O-Pins
XCS05	10 x 10	100	360	77
XCS10	14 x 14	196	616	112
XCS20	20 x 20	400	1120	160
XCS30	24 x 24	576	1536	192
XCS40	28 x 28	784	2016	205

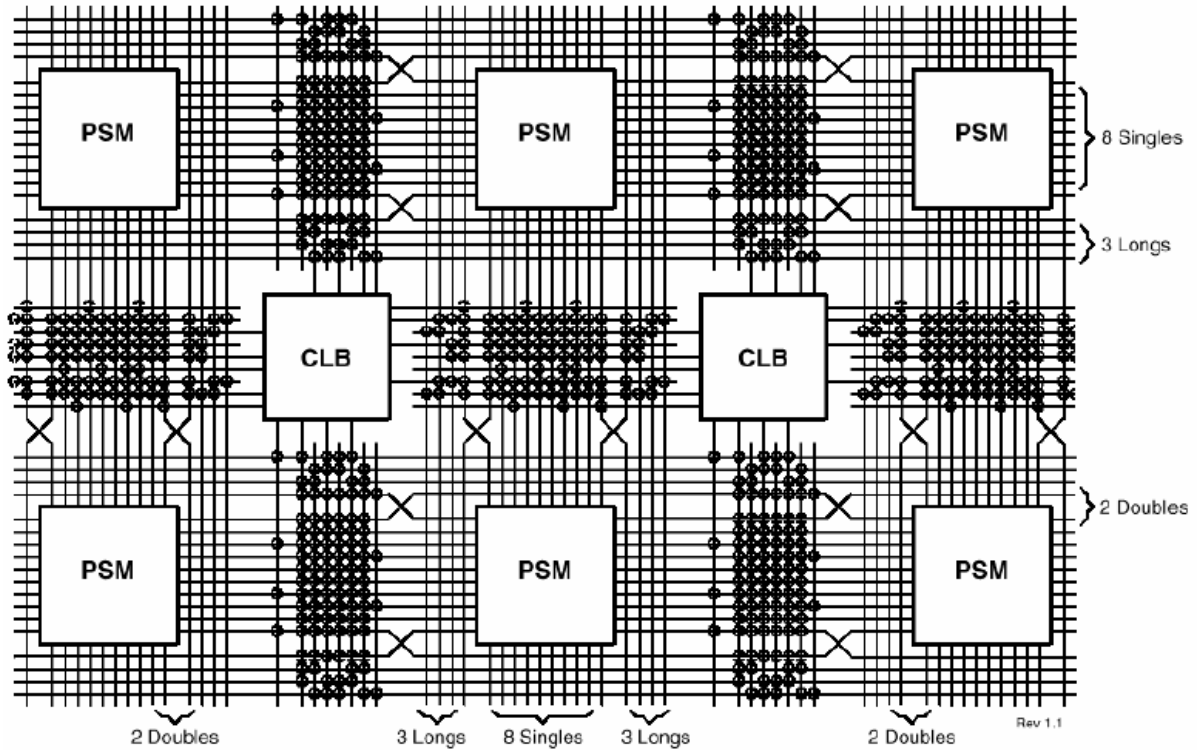
Bausteine benachbarter Größen gibt es meist in gleichen Gehäusen mit gleicher Anschlussbelegung, so daß eine Optimierung des Logikdesigns durch Austausch der Bausteine auch nach Erstellung des Platinenlayouts noch möglich ist.
Die CLBs der Spartan-Familie sind deutlich komplexer als die CLBs des Modell-FPGAs. Nachfolgend ist ein vereinfachtes Blockschaltbild eines CLB gezeigt.



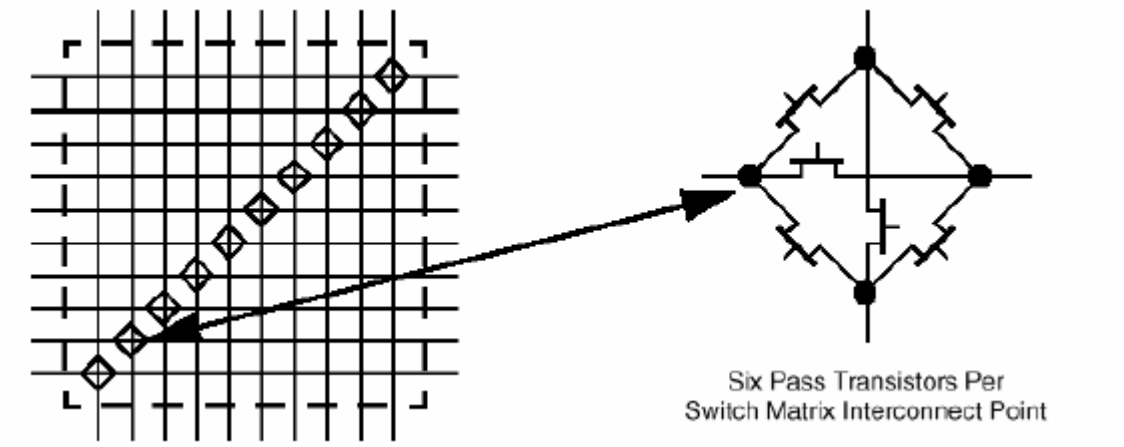
Jeder CLB besitzt drei Look-Up Tabellen. Die meisten Eingänge des CLBs sind auf die G-LUT und F-LUT geführt, mit denen jede beliebige Funktion aus den 4 Eingangssignalen realisiert werden kann. In einer zweiten Stufe sitzt die H-LUT, welche die Ergebnisse der ersten beiden LUTs mit einem Eingangssignal H1 verknüpfen kann. Jeder CLB kann somit beliebige Funktionen aus 5 Eingangsvariablen (H-LUT als Multiplexer zweier Funktionen F und G aus 4 Variablen) und manche Funktionen aus bis zu 9 Eingangsvariablen realisieren. Mittels programmierbarer Multiplexer können weiterhin unterschiedliche interne CLB-Funktionen auf die beiden Flip-Flops des CLBs und auf die Ausgangsleitungen gegeben werden.
Nachfolgend ist ein IO-Block gezeigt, links sind die Signale zum Inneren des FPGAs hin gezeigt, rechts ist der externe Anschlusspin zu sehen.



Jeder IO-Block enthält je ein Flip-Flop in Ein- und in Ausgangsrichtung, welches programmierbar in die Signalpfade eingeschaltet oder umgangen werden kann.
Am Ausgang ist ein Tri-State Treiber vorgesehen, der auch noch in seinem zeitlichen Verhalten programmiert werden kann. Weiterhin kann bei den Ausgangssignalen die Polarität programmierbar umgeschaltet werden.
Die nachfolgende Abbildung zeigt die Details der Routing Kanäle:



Man erkennt das zwischen den CLBs Routing-Leitungen verlaufen. Teils verbinden diese Leitungen benachbarte PSM-Elemente, ein Teil der Leitungen führt aber zum übernächsten PSM-Element. Erste werden als Single Length Lines, die zweiten als Double Length Lines bezeichnet. Damit verbessert man das Routing-Verhalten von Verbindungen mittlerer Länge. Für globale Verbindungen findet man weiterhin Long Lines, dies sind globale Leitungen über die Breite bzw. Höhe des ganzen Bausteins. Die Kombination dieser verschiedenen Routing-Leitungen deckt die verschiedenen Routing-Anforderungen ab und ermöglicht angepasste Verbindungen mit kurzem Delay.
Auf den Routing-Leitungen sind die PSP-Anschlusspunkte zu erkennen, über welche die CLBs an die Routing- Leitungen angeschlossen werden.
Die PSM-Elemente zum Zusammenschalten der Segmente aus Routing-Leitungen zu kompletten Signalleitungen besitzen die folgende Struktur:

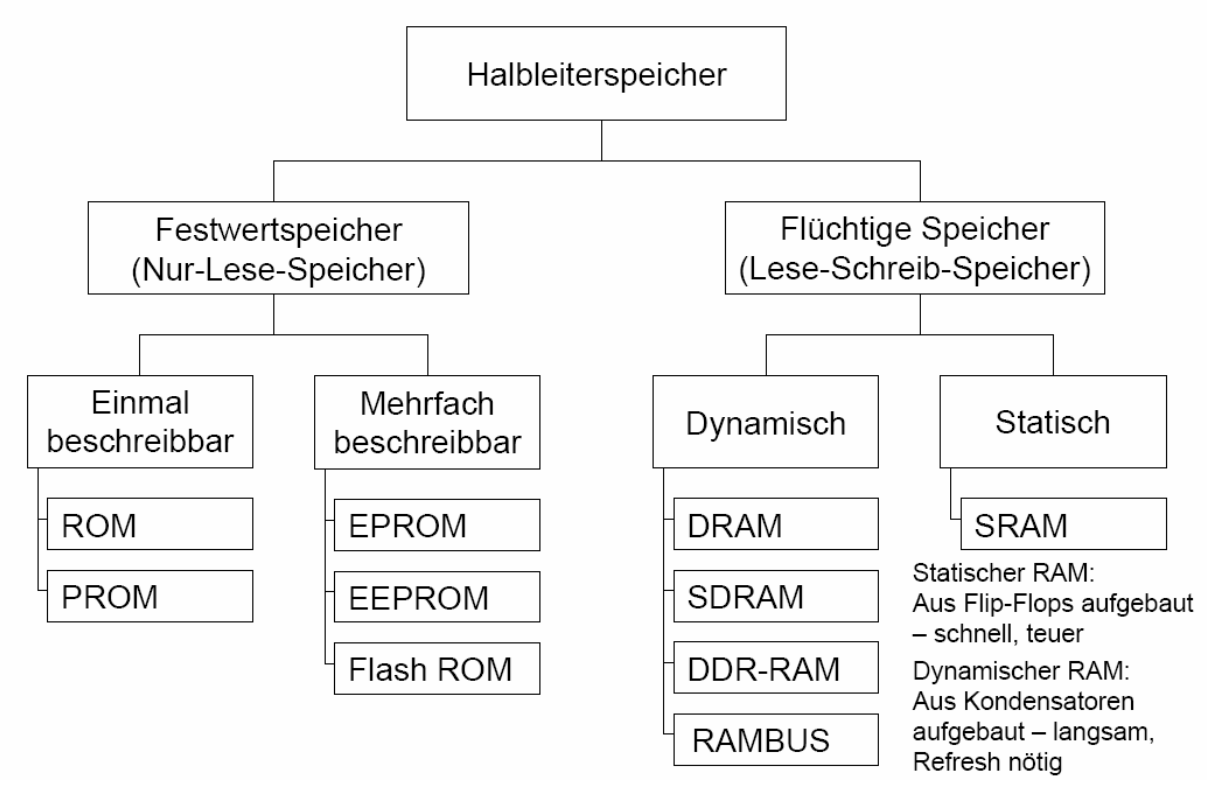


In einer diagonalen Linie, an der 4 Leitungssegmente zusammentreffen, sind 6 Schalttransistoren angeordnet. Diese erlauben das horizontale, vertikale und diagonale Zusammenschalten beliebiger Kombinationen dieser 4 Segmente.

15.1.2 Weitere Anbieter von FPGA-Bausteinen

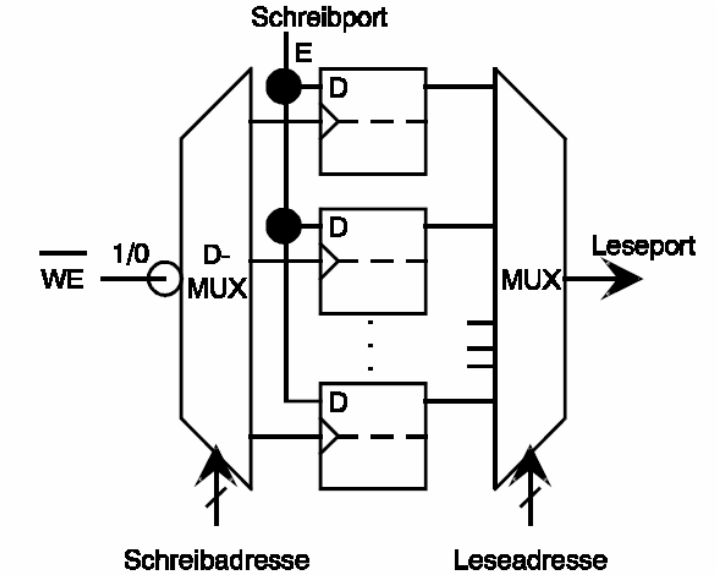
Weitere Anbieter von FPGA-Bausteinen sind beispielsweise die Firmen Altera (www.altera.com), Actel (www.actel.com), Atmel (www.atmel.com) und Quicklogic (www.quicklogic.com). Die FPGAs unterscheiden sich in der Struktur der Logikblöcke, der Anordnung der Routingkanäle etc. Allen Bausteinen ist jedoch gemeinsam, daß eine lokale, zellbasierte Struktur zum Aufbau der Logik verwendet wird.

16 Speicher



16.1 Aufbau von Speichern

Beim Entwurf von Schaltungen und Rechnern nehmen Speicher eine zentrale Stellung ein. Ein Speicher dient dazu, Daten aufzunehmen, zu speichern und wieder bereitzustellen. Daten können Worte mit fester Bitbreite oder auch Datenstrukturen (z.B. Zeichenketten) variabler Länge sein.
Die Auswahl einer Speicherzelle (Adressierung) erfolgt entweder explizit über einen Schlüssel (Adresse, Kennzeichnung) oder implizit über die Zugriffsreihenfolge.
Als Speicherelemente haben wir bereits Flip-Flops. Register und Registerfiles kennengelernt. Die prinzipielle Organisation eines Speicherelementes, das gleichzeitig das Laden und das Lesen gestattet, ist unten gezeigt. (Hier handelt es sich um einen hypothetischen Bitorientierten Zweiport-Speicher bestehend aus mehreren D-Flip-Flops). WE ist das Schreibsignal. Die Adressierung steuert das Schreiben (one-hot-Code); gelesen kann immer werden.

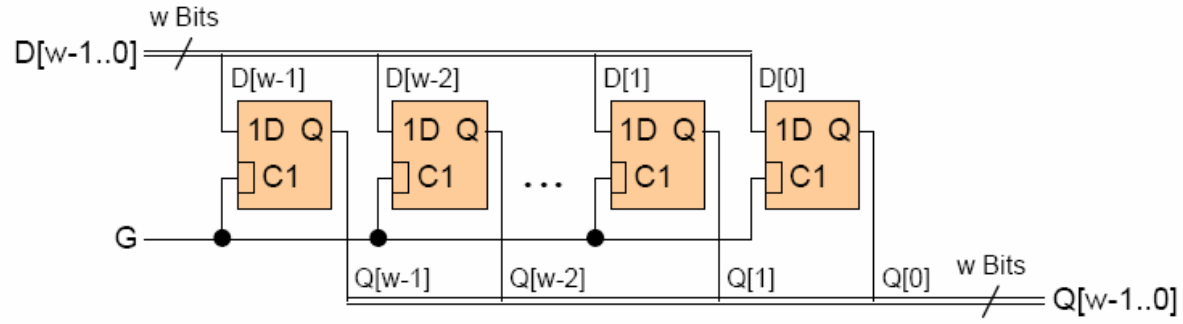


Größere Speicher bestehen aus Tausenden von Speicherzellen zusammen mit einer Zugriffslogik für den Transfer von Information in und aus den Speicherzellen. Wenn die Speicherzellen wahlfrei angesprochen werden können, nennt man einen derartigen Speicher einen RAM-Speicher (random access memory).

16.2 Speicherbausteine

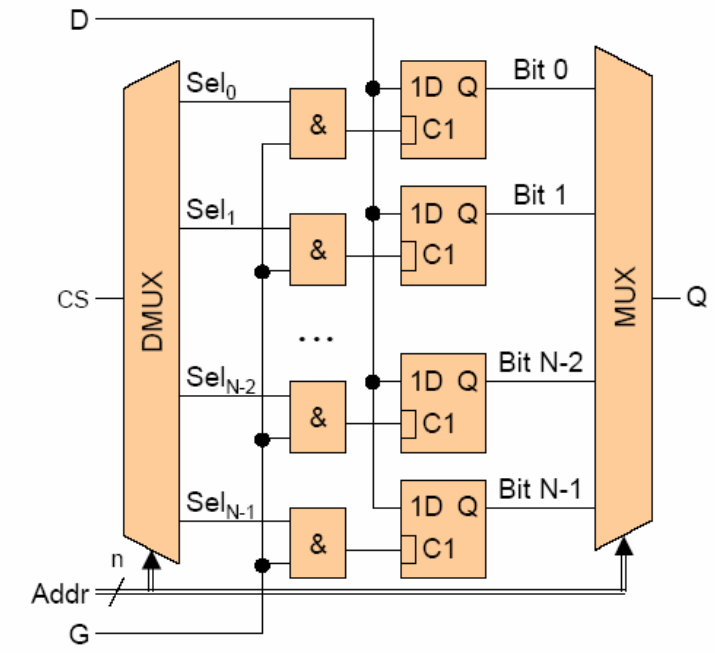
16.2.1 Statische Schreib-/Lesespeicher (RAM)

Mit RAM-Speicher (RAM-Random Access Memory) werden Speicher bezeichnet, welche ein Schreiben und Lesen von frei auswählbaren/adressierbaren Speicherworten ermöglichen. Bei ihnen ist die Zugriffszeit unabhängig vom Ort (Adresse) der Speicherung.
Grundelement aller statischen Speicher ist normalerweise eine 1-Bit Speicherzelle, die als pegelgesteuertes Flip-Fflop beschrieben werden kann.
Ein erster einfacher Speicher entsteht, wenn N=2n EinzelBits gespeichert werden müssen. Durch einen Adreßeingang G wird eine Speicherzelle am Eingang und am Ausgang ausgewählt. Aus dieser Speicherzelle können ganze Wortspeicher paralleler Bitspeicher aufgebaut werden (Register), deren einzelne Zellen durch Demultiplexer und Multiplexer auswählbar sind.

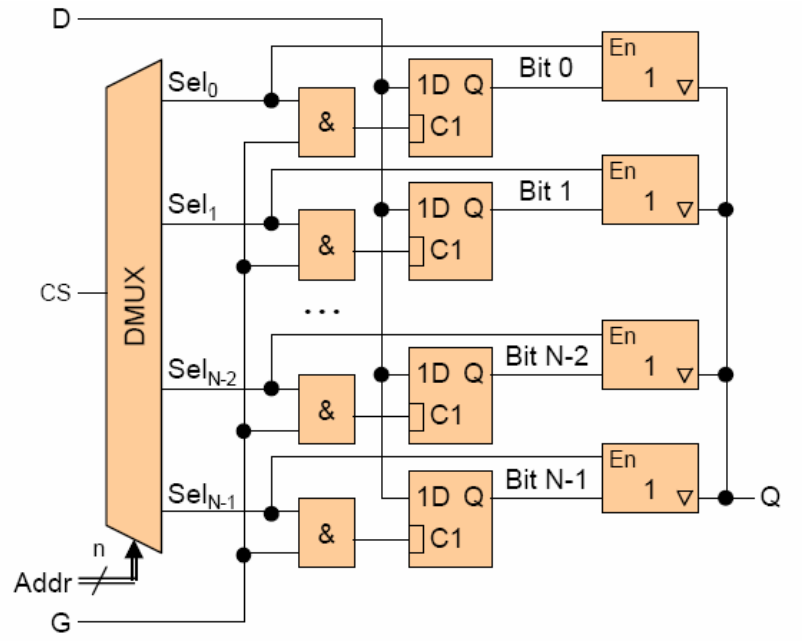


16.2.1.1 Speicherung von N EinzelBits

Es wird mit Hilfe eines über Addr angesteuerten Multiplexers das Datum der gewünschten Speicherzelle auf den Ausgang durchgeschaltet.



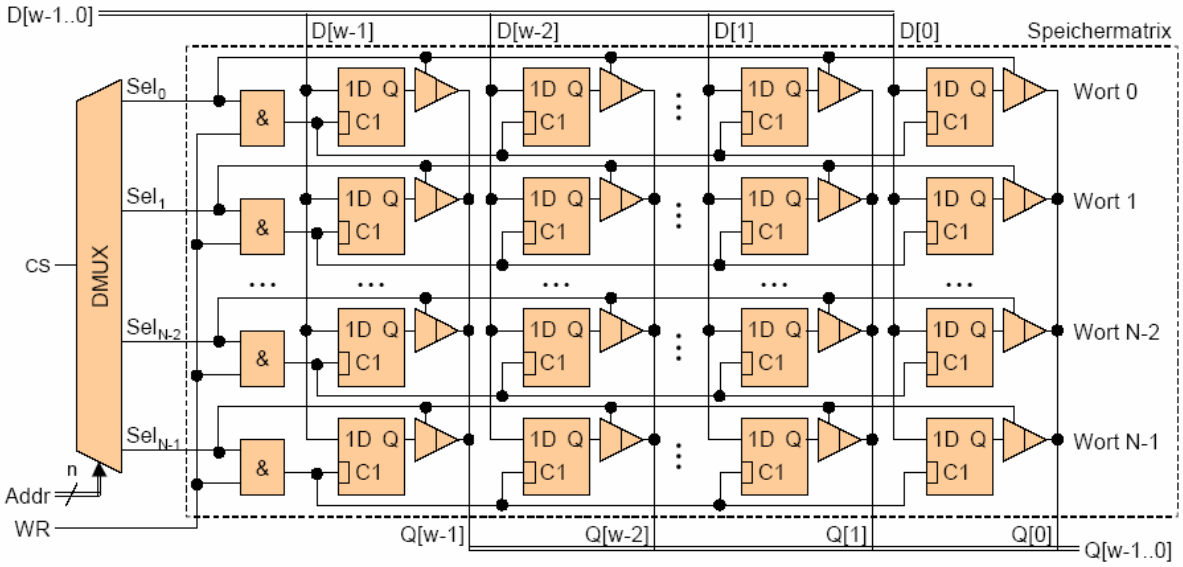
Am Eingang wird durch Addr mit einem Demultiplexer ein Selektionssignale SEL erzeugt, das zusammen mit einer generellen Freigabe (enable) eine Speicherzelle zum Schreiben bestimmt. Das zu schreibende Datum D liegt allen Zellen gemeinsam an.
Die Selektionssignale des Demux können in einer modifizierten Schaltung für die Ansteuerung von Tri-State Ausgängen verwendet werden.



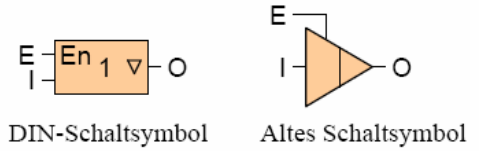
16.2.1.2 Speicherung von N Worten aus w Bits

Verbreitert man die Wortbreite von einem auf w Bits, so enthält jeweils eine horizontale Bank ein Wort zum Schreiben oder Lesen. Durch Ansteuerung mit einer Sel-Leitung werden dann

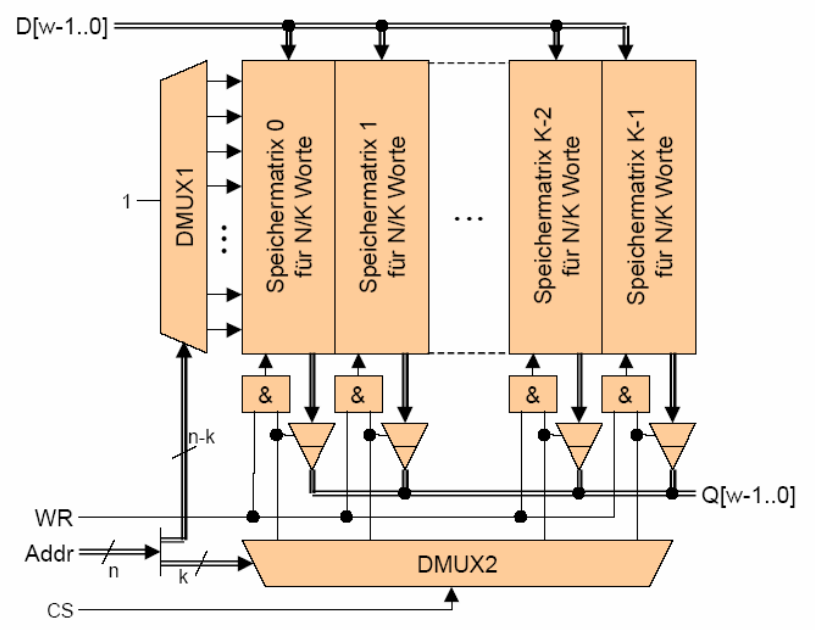
alle Bits eines Wortes angesprochen und mit G geschrieben. Im Moment noch ist immer einer der Ausgänge der Tri-State-Treiber aktiviert.



Tri-State-Treiber in alter und neuer Symbolik.



Funktional besitzt dieser Speicher schon die gewünschten Eigenschaften. Physikalisch würde für diese Architektur jedoch auf einem Halbleiter eine sehr schmale und hohe Fläche benötigt werden, da die Anzahl der Speicherworte eines Speicherbausteins (z.B. 220) um Größenordnungen größer ist als die Wortbreite eines Bausteins (z.B. 16=24). Durch die Zerlegung der schmalen, hohen Speicherfläche in 2k Speichermatrizen mit weniger Speicherworten (2n-k Speicherworte) lassen sich mehrere solcher kleineren Speichermatrizen nebeneinander anordnen, so daß die Anzahl der Speicherworte gleich bleibt und man trotzdem ein sinnvolles Seitenverhältnis erhält. Der Demultiplexer DMUX2 dient zur Selektion einer kleinen Speichermatrix, der Demultiplexer DMUX1 wählt innerhalb dieser Matrix die zu schreibende und zu lesende Speicherzelle aus. Die an die kleinen Speichermatrizen angeschlossenen Schreibsignale werden aus dem Schreibsignal WR des Gesamtspeichers erzeugt, so daß genau eine kleine Speichermatrix zum Schreiben ausgewählt wird. Ebenso wird der Ausgang genau einer kleinen Speichermatrix auf den Ausgang Q des Speichers geschaltet.

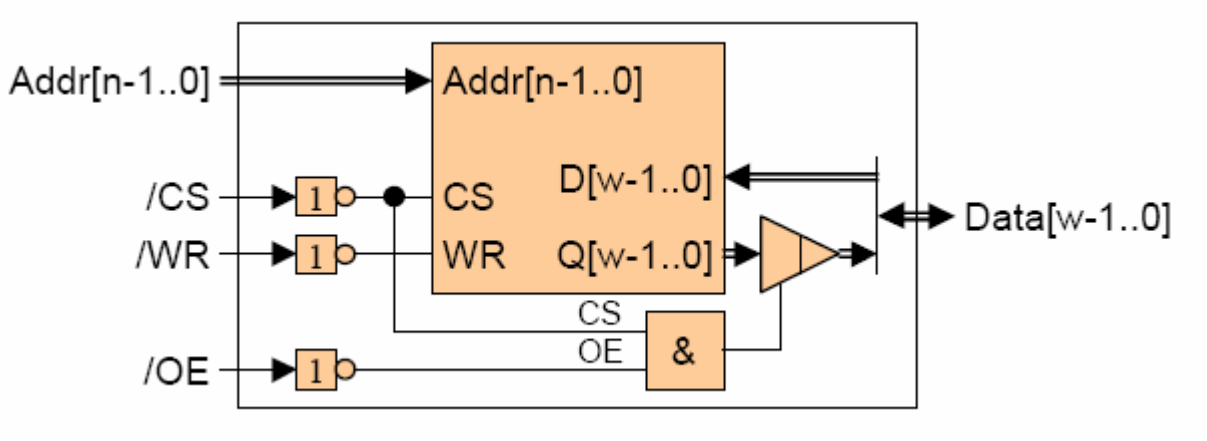


Speichermatrix mit K Spalten zu N/K Worte

Die Schaltung zur Generierung des Schreibsignals für die durch DMUX2 selektierte kleine Speichermatrix und zur Ausgabe des Ausgangssignals dieser Matrix auf den Speicherausgang wird in Analogie zum einfachen Speicher (siehe Abbildung) mit dem Demultiplexer DMUX2, mit UND-Gattern und Tri-State Treibern beschrieben.

16.2.1.3 Reale Speicherbausteine

Reale Speicherbausteine besitzen meist Steuereingänge (WR, CS) in invertierter Logik1. Dies bedeutet, daß beispielsweise die Auswahl eines Bausteins mit /CS=0 erfolgt.

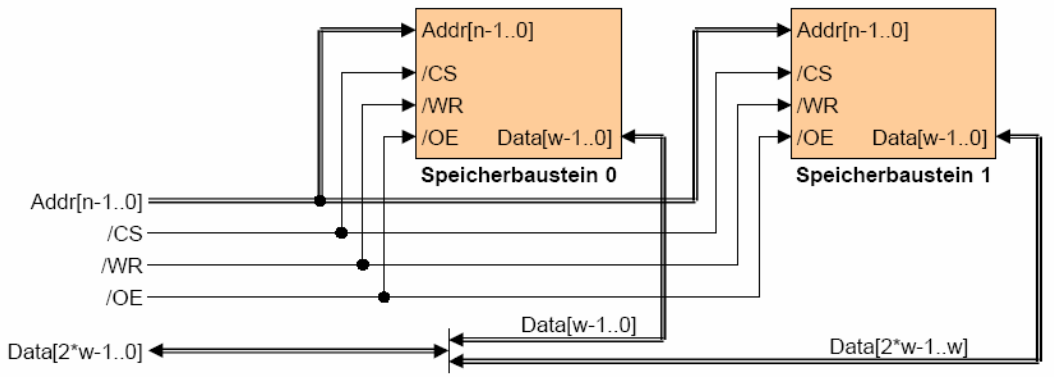


Werden Dateneingang und Ausgang im Multiplex auf demselben Pin geführt, so wird ein weiteres Steuersignal OE = output enable, benötigt, das den Tri-State-Zustand des Datenausgangs steuert.

Beim Lesen des Speichers wird das selektierte Speicherwort erst dann auf den Datenleitungen Data ausgegeben, wenn der Baustein ausgewählt ist (/CS=0) und der Baustein gelesen werden soll (/OE=0). Beim Schreiben des Speichers wird der Speicher durch /CS=0 selektiert, das Signal /OE=1 deaktiviert jedoch die Ausgabe von Daten durch den Speicher.

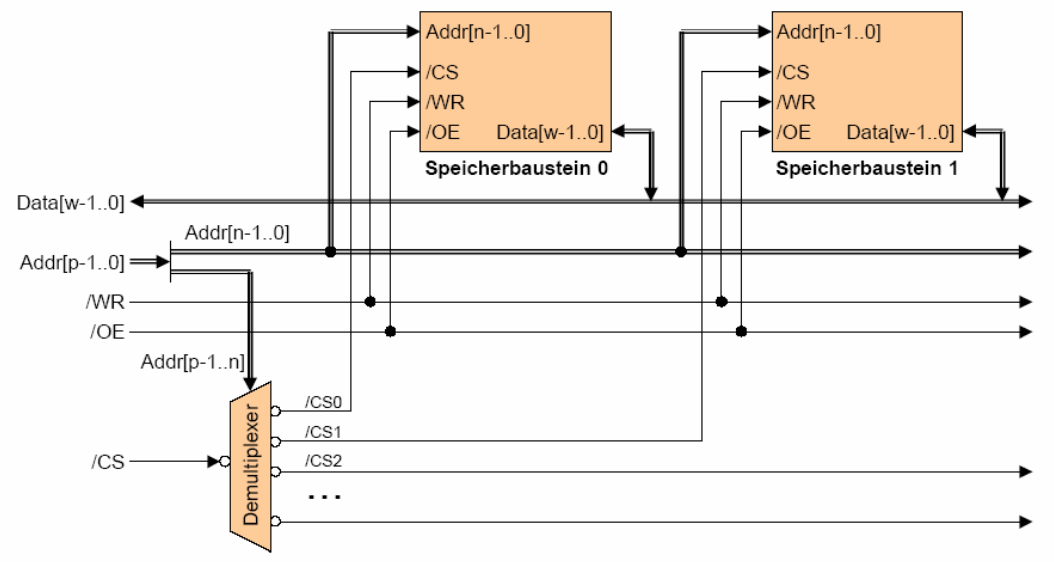
16.2.1.4 Vergrößern der Datenwortbreite

Werden in einem System Speicher einer Datenbreite w eingesetzt, jedoch doppelt breite Daten benötigt, so kann man zwei Bausteine nebeneinander kaskadieren und gemeinsam ansteuern. Jeweils ein Baustein liefert dann die halben Daten des gesamten Wortes.



16.2.1.5 Erhöhung der Anzahl der Speicherworte

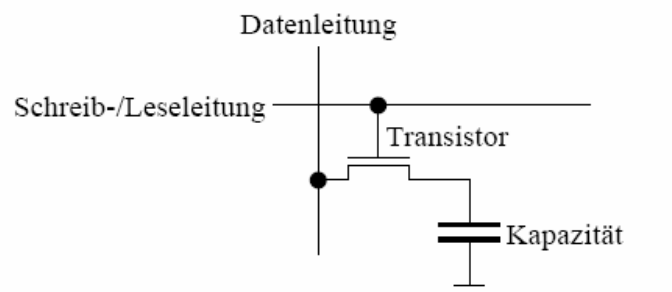
Soll die Speichertiefe vergrößert werden, so können die Speicher quasi übereinander kaskadiert werde. Dafür können WE, OE, ein Teil der Adressen und die Daten gemeinsam an beide Bausteine angeschlossen werden. Mit einem anderen Teil der Adressen und einem Demultiplexer werden Selektionsdaten erzeugt, mit denen ein Baustein aus der Speicherbank zum Lesen oder Schreiben selektiert wird .



Man beachte die negative Logik der CS-Signale.

16.2.2 Dynamische Schreib-/Lesespeicher (DRAM)

Elementares Speicherelement dynamischer Speicher ist eine Kapazität. Sie wird beim Schreiben auf einen dem Logikzustand zugeordneten Pegel geladen. Beim Lesen wird der Pegel abgefragt und als Logikzustand nach außen gegeben.
Zur Ansteuerung dient ein elektronischer Schalter, der über eine Schreib-/Leseleitung gesteuert ist und die Kapazität mit einer Datenleitung verbindet. Die Realisierung des Schalters erfolgt mit einem Transistor, die Kapazität wird mit geeigneten Halbleiterstrukturen realisiert



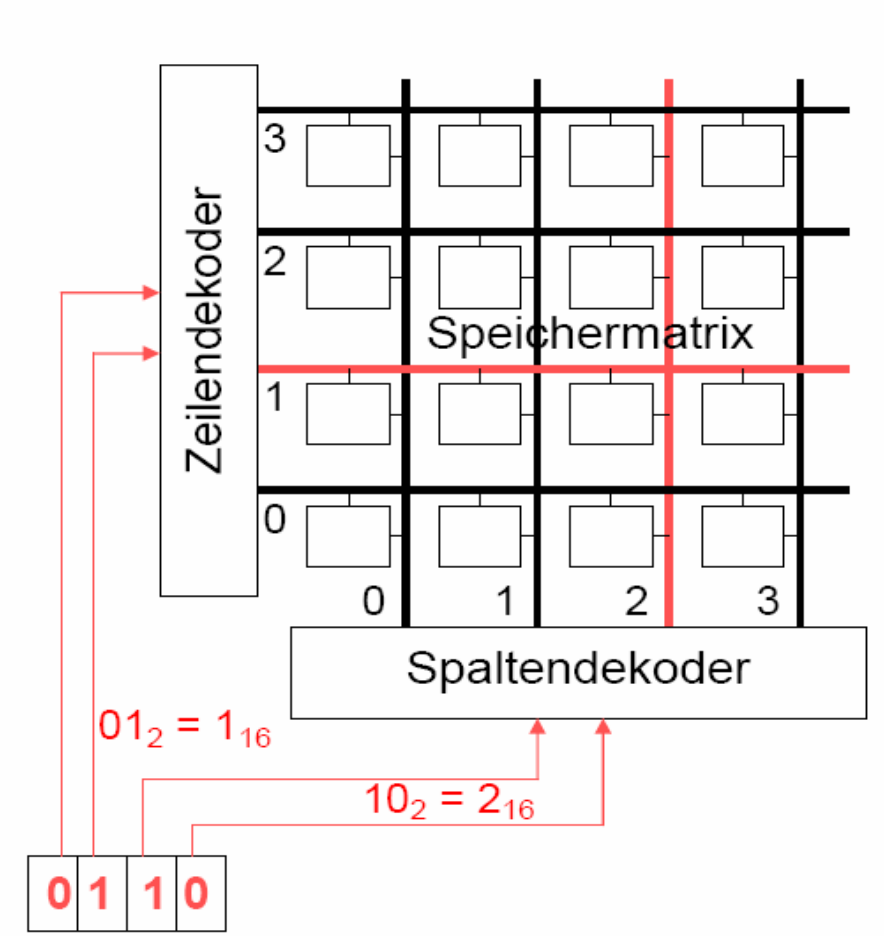
Soll eine 1 gespeichert werden, so wird die Kapazität über die Datenleitung aufgeladen, bei Speicherung einer 0 wird die Kapazität entladen. Beim Lesen der Zelle entlädt sich die Kapazität, sie muß also anschließend wieder durch Rückschreiben wieder aufgeladen werden.

16.2.2.1 Refresh: Auffrischen der Information

Die Kapazität einer Speicherzelle ist nicht perfekt. Daher erfolgt mit der Zeit eine langsame Entladung der Zelle. Um einen Informationsverlust zu vermeiden, muß die Information der Zelle in regelmäßigen Abständen (ca. 2 bis 16 ms) aufgefrischt werden. Dazu wird die Information aus der Zelle gelesen und sofort wieder eingeschrieben. Beim Einschreiben erfolgt ein vollständiges Laden oder Entladen der Kapazität, je nachdem ob eine 1 oder eine 0 eingeschrieben war. Beim Refresh bleibt somit der Wert der Zelle unverändert, der Spannungspegel der Kapazität wird jedoch wieder auf einen optimalen Wert gesetzt.

16.2.2.2 Architektur dynamischer RAM-Bausteine

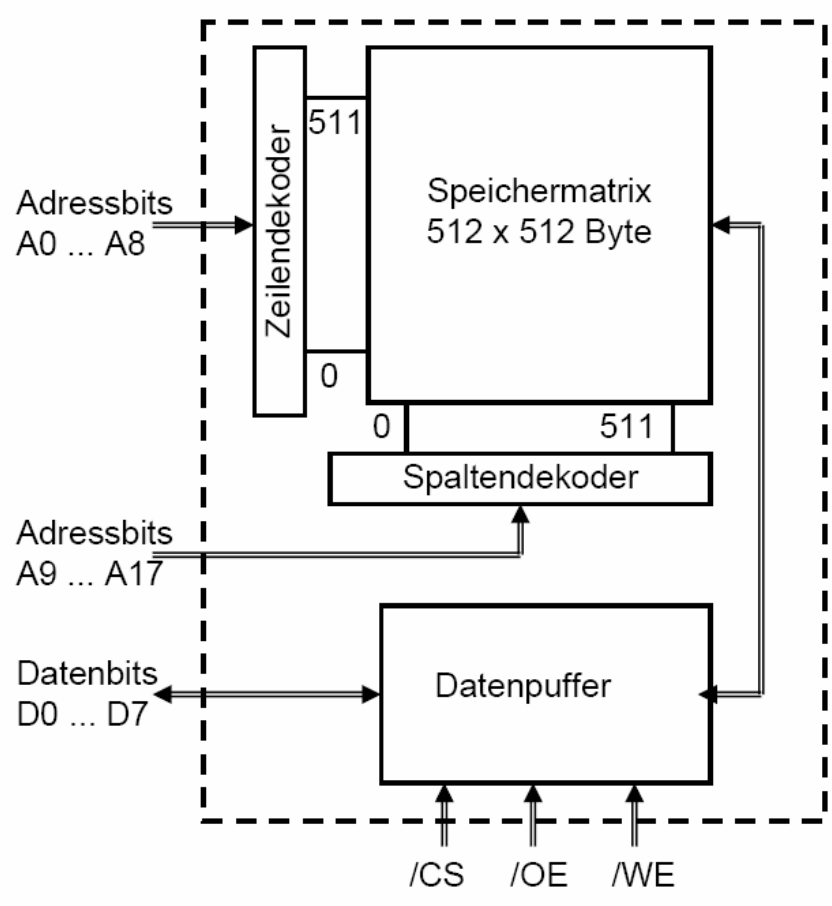
In einem dynamischen Speicherbaustein werden viele dynamische RAM-Zellen in einer Matrix Zeilen- und Spaltenweise angeordnet.

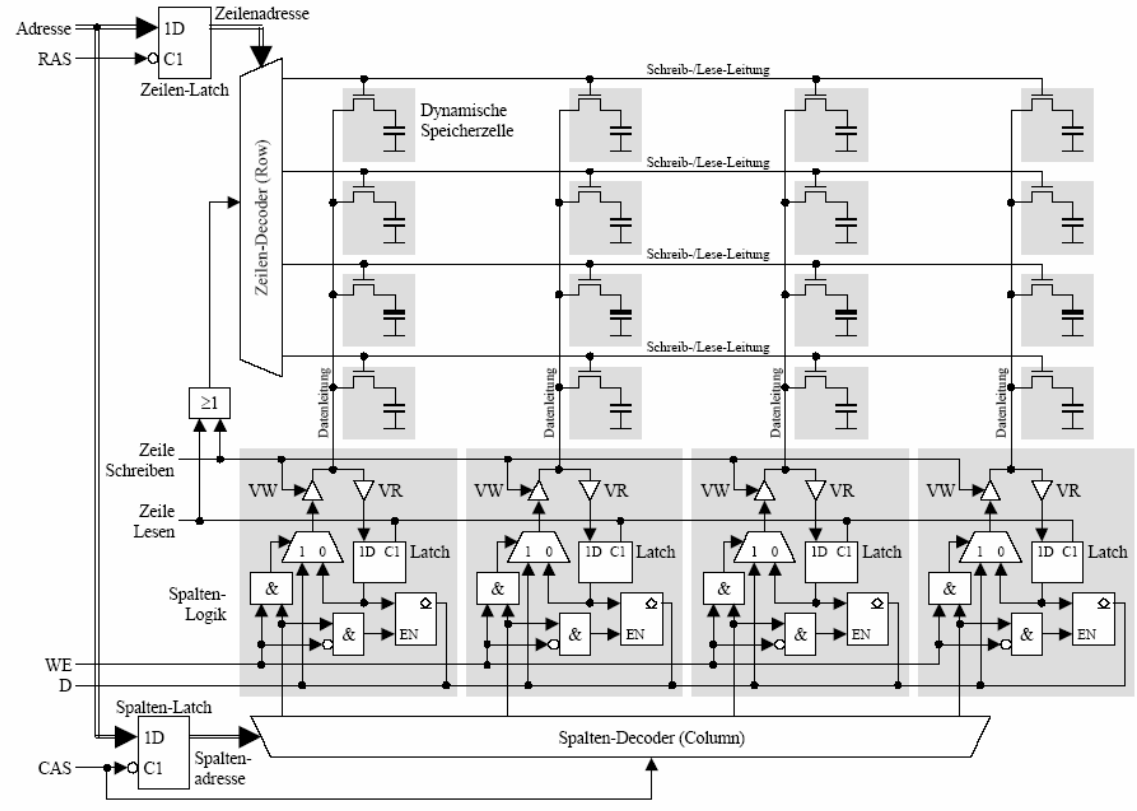


Der Speicher im Bild hat vier Adressleitungen und es können also $2^4 = 16$ Zellen angewählt werden.
Speicherzelle selektieren:
Adresse teilen und
• einen Teil als Spaltenadresse und den
• anderen Teil als Zeilenadresse interpretieren
Die Adressdekoder (Zeilendekoder und Spaltendekoder) wählen entsprechend der anliegenden Adresse 0110 die Zeile 1 und die Spalte 2 aus der Speichermatrix aus.

Beispiel: 18 Bit Adressen und 8 Bit Daten Steuerleitungen:
Die Steuerleitungen arbeiten meist mit negativer Logik (z.B. /CS).
CS - Chip Select: Mit dem Steuersignal CS wird festgelegt, ob der Datenpuffer für den außen anliegenden Datenbus transparent ist oder nicht. Bei CS=1 ist der Datenpuffer gesperrt.
OE - Output Enable: Signalisiert in Zusammenhang mit CS einen Lesevorgang.
WE - Write Enable: Signalisiert in Zusammenhang mit CS einen Schreibvorgang.

Tatsächlich werden die Adressen über einen gemeinsamen Adreßbus beiden Decodern gleichzeitig zugeführt. Mit den sogenannten RAS und CAS Signalen wird gesteuert, welche Bedeutung die gerade angelegte Adresse hat und welcher Decoder diese übernehmen muß, siehe unten.





Jede Zeile der Matrix besitzt eine gemeinsame Leseleitung, die über eine Zeilenadresse ausgewählt wird.

Die Zeilenadresse wird von außen an den Speicher angelegt und mit dem RAS-Signal (Row Adress Select) in ein pegelgesteuertes Register übernommen. Bei Auswahl einer Zeile werden alle angesprochenen RAM-Zellen mit den zugehörigen Datenleitungen verbunden.
Alle Zellen einer Spalte besitzen eine gemeinsame Datenleitung. Zu einem Zeitpunkt wird jedoch nur eine einzelne Zelle der Spalte durch die Zeilenadresse ausgewählt und damit an die gemeinsame Datenleitung angeschaltet.
An jede Datenleitung ist eine Hardware zum Lesen und Schreiben der ausgewählten Zelle angeschaltet, sie ist als Spaltenlogik bezeichnet. Das Schreiben und Lesen der Zellen erfolgt immer Zeilenweise.
Zum Lesen der durch die Zeilenadresse ausgewählten RAM-Zeile werden die Spannungspegel der Kondensatoren in Logikpegel gewandelt. In der Prinzipschaltung sind dafür die Verstärker VR vorgesehen. Die Logikpegel werden in Latches eingelesen und steht dort für die weitere Verarbeitung zur Verfügung. Die Kondensatoren der ausgewählten Zeile haben sich durch das Auslesen entladen (destruktives Lesen). Zum Erhalten der Nachricht muß also der ursprüngliche Inhalt der Zeile wieder in die Zellen eingeschrieben werden.

Zum Schreiben einer Zeile werden die Werte über den Verstärker VW auf die Zellen der ausgewählten RAM-Zeile gegeben und damit die Kondensatoren der Zellen aufgeladen. Die Werte werden den Latches entnommen. Für die durch die Spaltenadresse ausgewählte Spalte wird bei aktivem WR-Signal jedoch der Wert der externen Datenleitung D in die RAM-Zelle eingeschrieben. Die Spaltenadresse wird über die gleichen Adressleitungen wie die Zeilenadresse an den Speicher angelegt und mit dem CAS-Signal (Column Adress Select) in ein pegelgesteuertes Register übernommen.

Die Signale zum Lesen und Schreiben der Zeilen werden intern im Speicher aus den RAS- und CAS-Signalen erzeugt.

Die Operationen zum Lesen und Scheiben einzelner Speicherzelle müssen die vorgestellten Operationen zum Lesen und Schreiben einer Speicherzeile verwenden.

16.2.2.3 Lesen:

Zum Lesen einer Zelle werden mittels RAS- und CAS-Signal die Zeilen- und Spaltenadresse in den Speicher eingeschrieben. Der Speicher liest dann eine ganze Zeile in die Latches der Spaltenlogik ein. Da durch diesen Lesevorgang der Inhalt der RAM-Zellen zerstört wird, führt der Speicher anschließend einen Schreibvorgang durch, der den Inhalt der Zeile wieder in die Speicherzellen einschreibt. Von allen in den Latches gespeicherten Werten wird schließlich der durch die Spaltenadresse adressierte Wert über die Datenleitung D nach außen gegeben.

16.2.2.4 Schreiben:

Beim Beschreiben einer Zelle werden zunächst wieder die Zeilen- und Spaltenadresse eingeschrieben, anschließend eine gesamte Speicherzeile gelesen und in den Latches der Spaltenlogik gespeichert. Beim nachfolgenden Schreibvorgang wird in der durch die Spaltenadresse adressierten Spalte der Wert der Datenleitung D eingeschrieben, in allen übrigen Spalten werden die in den Latches gespeicherten Werte zurückgeschrieben. Während des Einschreibens muß das Schreibsignal WE aktiviert sein.

16.2.2.5 Refresh:

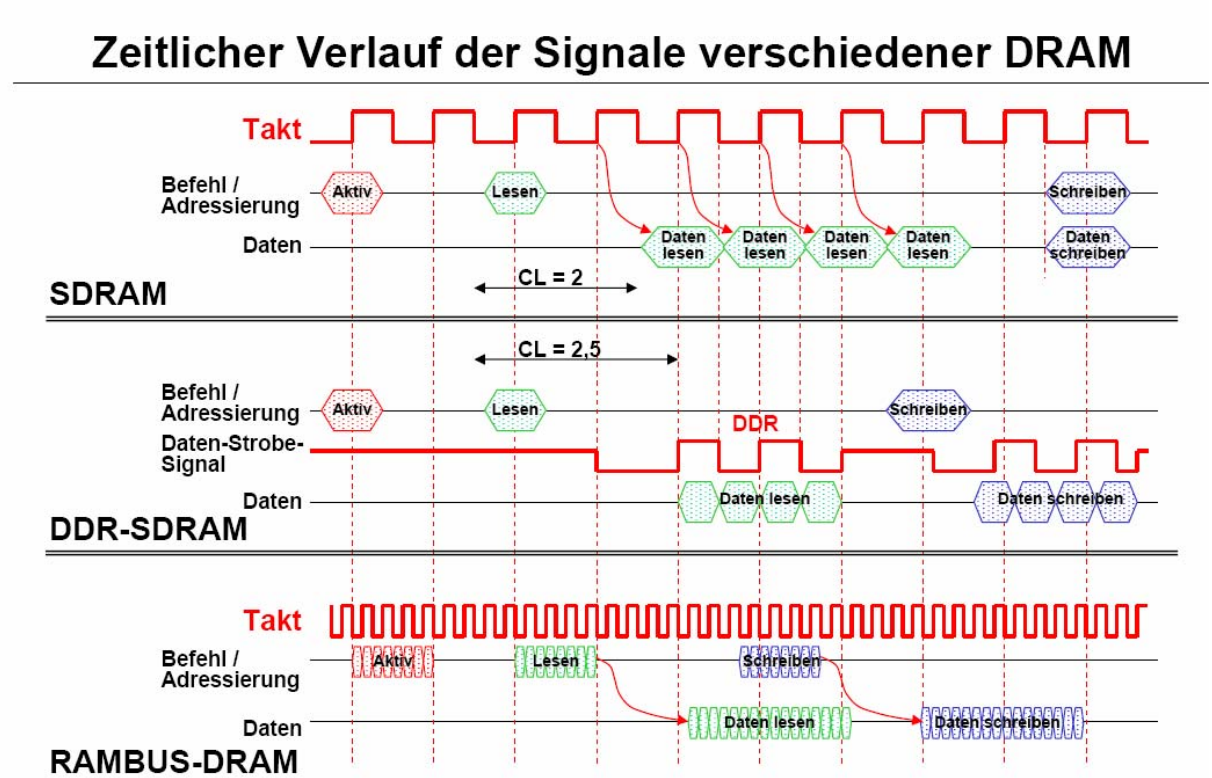
Bei jedem Lese- und Schreibvorgang erfolgt zunächst ein destruktives Lesen und neues Beschreiben der ausgewählten Zeile. Somit werden alle Zellen einer Zeile dabei aufgefrischt. Es gibt verschieden Refresh-Modi:
Ras only Refresh: Typischerweise bieten dynamische Speicher noch einen zusätzlichen Zyklus an, bei dem nur das Lesen und Schreiben der ausgewählten Speicherzeile durchgeführt wird. Für diesen Vorgang wird nur die Zeilenadresse an den Speicher angelegt.
Im Hidden Refresh wird ein normaler Schreib/Lesezyklus etwas verlängert und in dieser Zeit weitere Zellen refreshed. Dafür adressiert ein interner Refresh-Zähler die Zellen. Der gleiche Zähler findet beim Self Refresh Anwendung, selbst wenn der ansteuernde Prozessor abgeschaltet ist, können sich die RAMs selbst refreshen. Auch beim Cas before Ras Refresh wird der eigene Zähler ausgenutzt.
Da bei allen Zugriffen nicht nur eine Zelle, sondern eine ganze Speicherbank angesprochen wird, ist auch der Refresh nur pro Speicherbank erforderlich.

16.2.2.6 Beschleunigung des Zugriffs

Page/Nibble Mode: Beim beschriebenen Vorgehen zum Lesen und Schreiben eines Datenwortes wird jedes Mal eine vollständige Zeile aus dem Speicher gelesen und wieder eingeschrieben. Beim Lesen von Werten innerhalb einer Speicherzeile braucht nur beim ersten Zugriff auf die Zeile der Inhalt in die Latches gelesen werden. Anschließend kann allein über die Spaltenadresse auf die in den Latches gespeicherten Werte zugegriffen werden. Der langsame Zugriff auf die Speicherzeilen ist bei den Folgezugriffen nicht mehr notwendig. Entsprechendes gilt beim Schreiben, wenn bei jedem Schreibvorgang der Inhalt des Latches der ausgewählten Spalte korrigiert wird.

16.2.2.7 Aktuelle RAM Typen

SDRAM (Synchronous Dynamic RAM) wird mit einem Taktgeber synchronisiert, der vom CPU-Takt abgeleitet ist. Alle Schreib- und Lesevorgänge werden von der steigenden Flanke dieses Taktes ausgelöst.
DDR-SDRAM (Double Data Rate) ist eine schnellere Variante des SDRAM. Die Daten werden mit der steigenden und der fallenden Taktflanke gelesen oder geschrieben, wodurch sich die Datenrate des Speicherchips verdoppelt. Dazu wird der in einem Chip befindliche Speicher in mehrere Bänke aufgeteilt, die abwechselnd ausgelesen werden (Speicherverschränkung).
RDRAM ist eine spezielle Entwicklung der Firma Rambus. Daten werden aus vielen Bänken parallel gelesen und dann über einen Multiplexer auf den Datenausgang schaltet. Z.B. können aus 8 Bänken gleichzeitig 8 Byte gelesen werden. Das entspricht dann der 8-fachen internen Lesegeschwindigkeit.



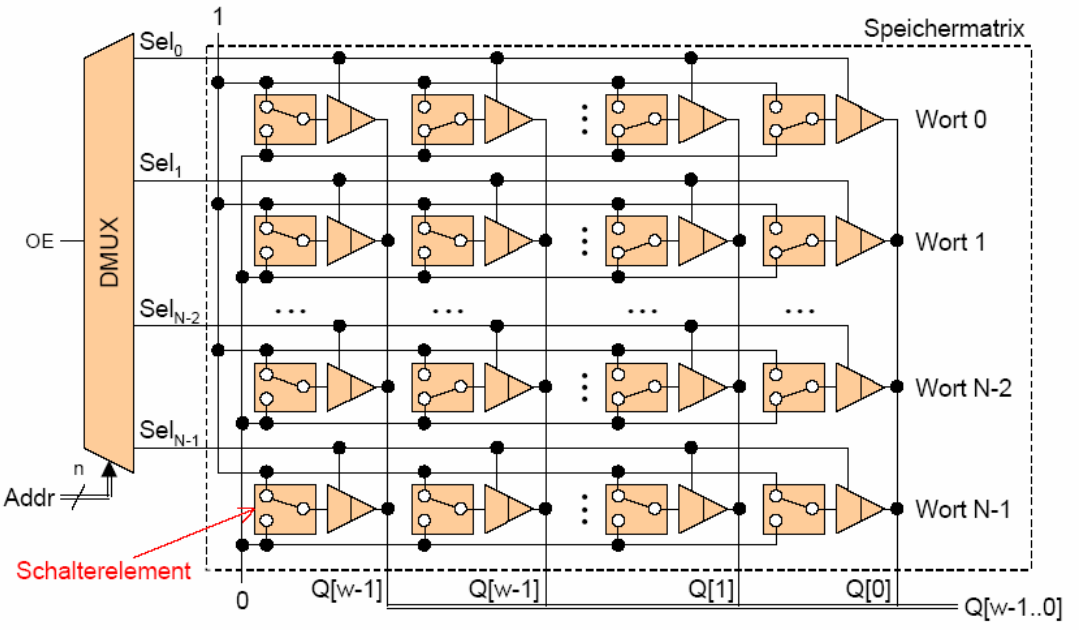
CAS (Column Address Strobe) in Spaltenadresspuffer
RAS (Row Address Strobe) in Zeilenadresspuffer
CL = CAS Latency

Voraussichtliche Entwicklung der DDR-Speicher			
DDR-RAM Typ	Takt-frequenz	Band-breite	Einfüh-rung im Jahr
PC 5400	333 MHz	5,5 GB/s	2005
PC 4300	266 MHz	4,3 GB/s	2004
PC 3200	200 MHz	3,2 GB/s	2003
PC 2700	166 MHz	2,7 GB/s	2002
PC 2100	133 MHz	2,1 GB/s	2000

Berechnungsbeispiel:
133 MHz x 8 Byte x 2 Zugriffe/Takt
➔ 2,1 GB/s

16.2.3 Lesespeicher (ROM)

Ein ROM-Speicher (Read-Only Memory) entspricht in der Struktur den vorgestellten statischen RAMSpeichern, er wird jedoch im normalen Betrieb nicht beschrieben. Daher kann man im Modell statt einer Speicherzelle einen Schalter vorsehen, der fest mit 1 oder mit 0 verbunden ist. Man erkennt, daß die Schaltungsteile zum Schreiben entfernt sind und somit nur noch ein Lesen möglich ist. Beim Lesen verhält sich der Speicher wie der statische RAMSpeicher.



16.2.3.1 ROM-Speicher

unterscheiden sich darin, wie die Schalterelemente realisiert werden. Dies führt zu unterschiedlichen Verfahren zur Programmierung der Schalterelemente. Folgende ROM-Typen sind gebräuchlich:

16.2.3.2 PROM (Programmable ROM)

Der Inhalt der Speicherzellen kann über Steuereingänge einmal programmiert werden.

16.2.3.3 EPROM (Erasable PROM)

Der Inhalt der Speicherzellen kann über Steuereingänge programmiert werden. Mittels UV-Licht ist der Speicher löschbar und kann anschließend neu programmiert werden.

16.2.3.4 EEPROM (Electrical Erasable PROM)

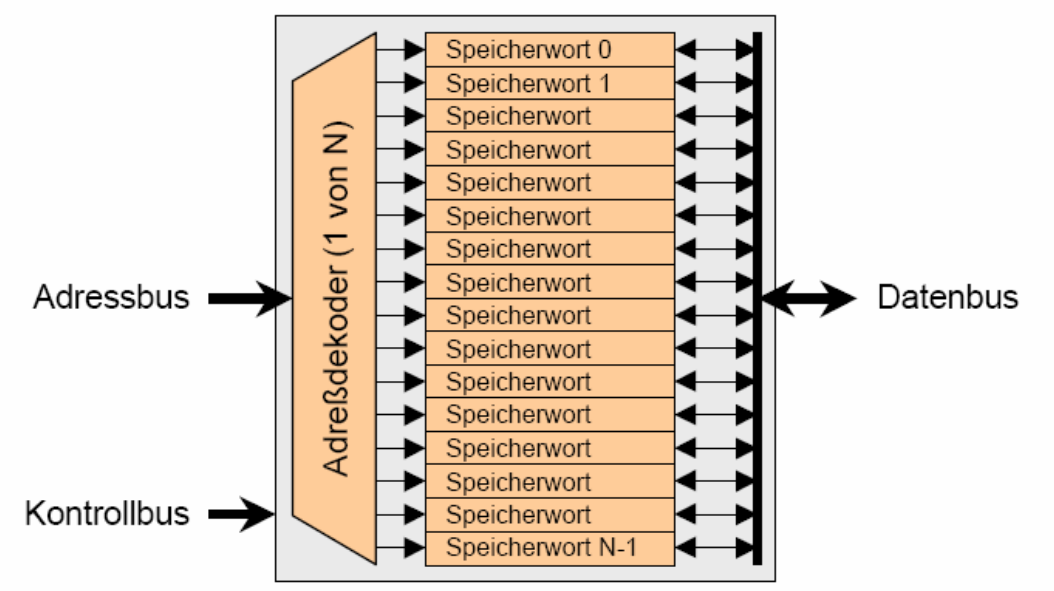
Der Inhalt der Speicherzellen kann über Steuereingänge programmiert werden. Ebenso kann der Speicher über diese Steuereingänge gelöscht und anschließend neu programmiert werden. Zu dieser Kategorie gehören auch sogenannte Flash-Speicher.

16.2.4 Speicherarchitekturen

16.2.4.1 Datenspeicher mit expliziter, direkter Adressierung

Der Zugriff auf ein Speicherwort erfolgt explizit und direkt über eine Adresse. Die Worte des Speichers sind in einem festen Adreßbereich ansprechbar, dieser Adreßbereich weist keine Lücken auf. Beim Zugriff auf den Speicher wird die angelegte Adresse als Binärzahl interpretiert. Die Adresse wird dekodiert nach der „1 von N“-Dekodierung. Dies bedeutet, daß

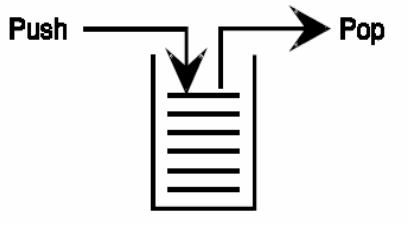
die Adresse einen Wertebereich von 0 bis N-1 besitzt und somit aktuell einen von N möglichen Werten repräsentiert. Dieser aktuelle Wert adressiert bei diesem Speichertyp genau ein Speicherwort. Der Zugriff über Kontroll- und Datenbus erfolgt dann auf das über die Adresse ausgewählte Wort.



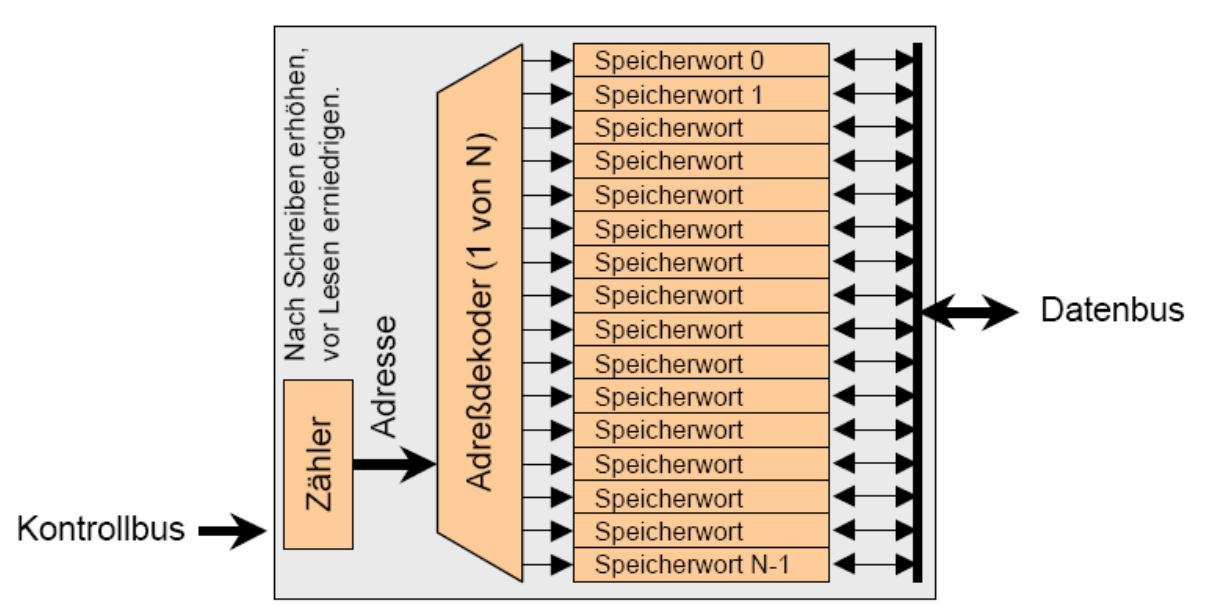
Dieser Speichertyp findet sich mit fester Datenwortlänge z.B. als Hauptspeicher in PC-Rechnern.

16.2.5 Stapelspeicher (implizite Adressierung)

Der Name „Stapelspeicher/Stack“ rührt daher, daß die in den Speicher eingeschriebenen Werte in den Speicherworten „aufeinandergestapelt“ werden. Beim Lesen ist, wie bei einem Stapel, nur der Zugriff auf das oberste Element möglich. Werden „tieferliegende“ Elemente benötigt, müssen alle darüberliegenden Werte zuvor gelesen werden.
Beim Stapelspeicher (Last In First Out (LIFO), Stack) erfolgt die Adressierung implizit über einen Auf- und Abwärtszähler. Der Zähler gibt die Adresse des zu schreibenden oder zu lesenden Speicherworts an, er zeigt genau auf das nächste freie Speicherwort.



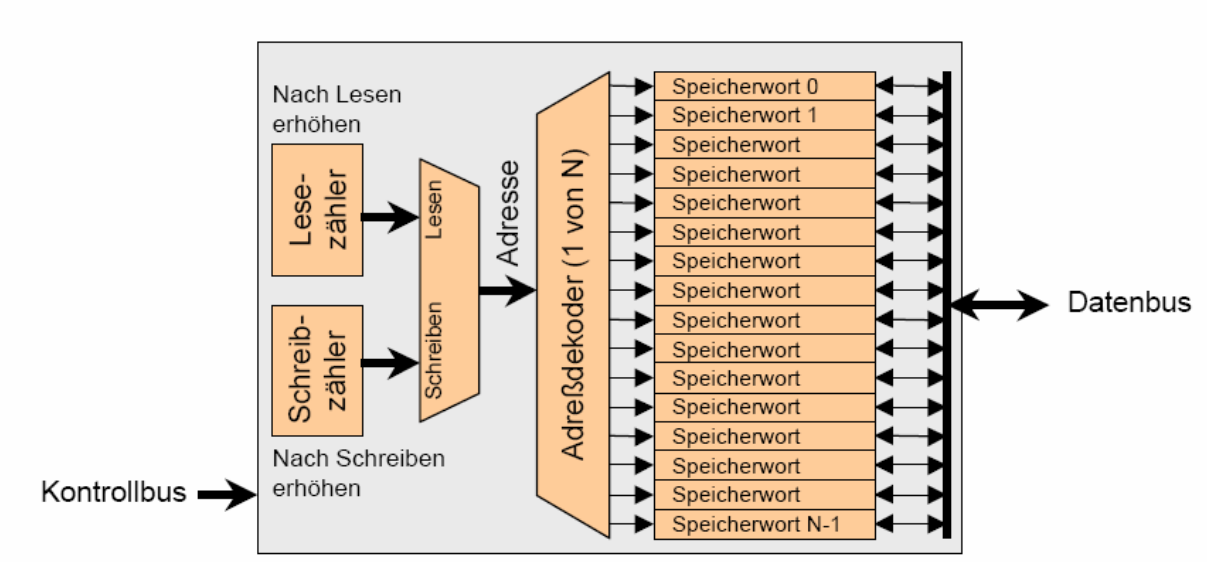
Beim Schreiben (Push) wird der Inhalt des Datenbusses in das vom Zähler adressierte freie Speicherwort geschrieben; anschließend wird der Zähler erhöht und zeigt damit auf das nächste freie Speicherwort. Bei aufeinanderfolgenden Schreibzyklen werden somit die Daten in aufeinanderfolgende Speicherworte mit aufsteigenden Adressen geschrieben.
Beim Lesen (Pop)wird zunächst der Zähler erniedrigt und dann der Inhalt des Speicherworts, welches durch den Zähler adressiert wird, auf dem Datenbus ausgegeben. Es werden beim Lesen somit die zuletzt geschriebenen Werte zuerst ausgelesen.



Ein Problem beim Stapelspeicher ergibt sich aus seiner endlichen Größe. Beim realen Einsatz ist zu klären, wie eine Überlaufsituation, d.h. der Zähler zeigt hinter das höchste Speicherwort und es wird ein Schreibzugriff durchgeführt, behandelt wird. Ebenso ist die Behandlung von Unterlaufsituationen, d.h. bei leerem Stapel (Zähler steht auf Null) wird ein Lesezugriff auf den Speicher durchgeführt, zu klären. In HW gibt es Vorrichtungen, diese Situationen vorher anzukündigen, z.B. durch Füllstandssignale, die fast voll oder fast leer signalisieren. Stapelspeicher werden bei Rechnern überall dort eingesetzt, wo temporär Daten zwischengespeichert werden und die Reihenfolge der Daten beachtet werden muß.

16.2.6 Schlange,queue (implizite Adressierung)

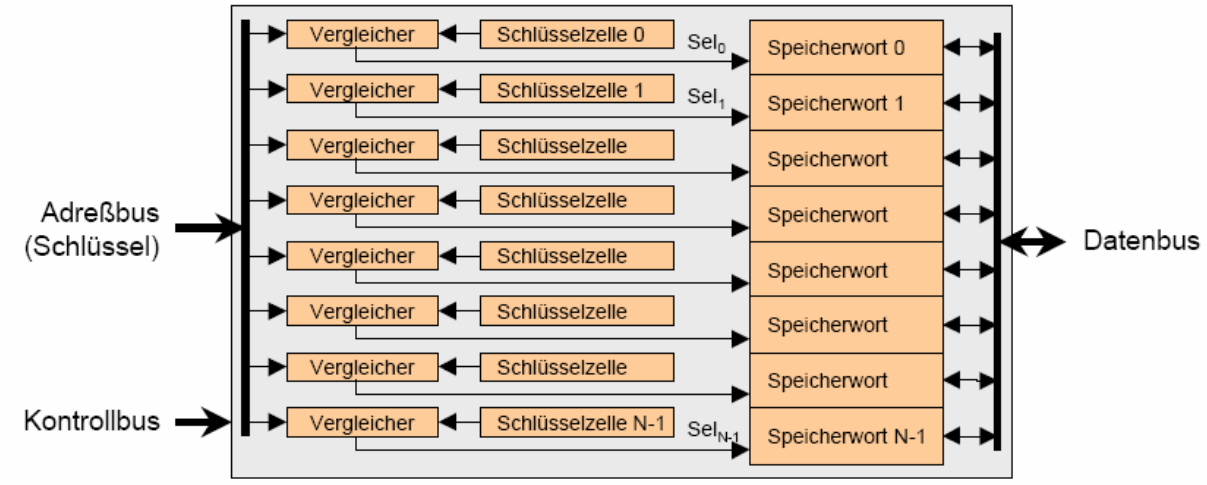
Der Name „Schlange“ rührt daher, daß bei diesem Speicher die zuerst eingeschriebenen Werte zuerst ausgelesen werden. Die Daten reihen sich somit in eine Schlange ein. Beim Schreiben können neue Werte nur am Ende der Schlange angestellt werden, beim Lesen nur Werte vorne am Beginn der Schlange entnommen werden.
Bei der Schlange (First In First Out (FIFO), Queue) erfolgt die Adressierung implizit über zwei Aufwärtszähler. Der eine ist Schreibzähler, er hält die Adresse des nächsten freien Speicherworts, in welches geschrieben werden darf, und zeigt somit direkt hinter das Ende der Schlange. Der zweite Zähler ist Lesezähler, er hält die Adresse des Speicherworts, welches als nächstes gelesen wird. Er zeigt somit auf den Beginn der Schlange.
Beim Schreiben wird der Inhalt des Datenbusses in das vom Schreibzähler adressierte freie Speicherwort geschrieben. Anschließend wird der Zähler erhöht, er zeigt damit auf das nächste freie Speicherwort. Die Daten werden somit in aufeinanderfolgende Speicherworte mit aufsteigenden Adressen geschrieben.
Erreicht der Schreibzähler die höchste Adresse, wird er beim nächsten Schreibzyklus zu Null gesetzt und die Speicherworte werden wieder bei Adresse 0 beginnend beschrieben. Holt der Schreibzähler den Lesezähler ein, so ist der Speicher voll und es dürfen ohne vorherige Lesezyklen keine Daten mehr in den Speicher eingeschrieben werden.
Beim Lesen wird der Inhalt des vom Lesezähler adressierten Speicherworts auf dem Datenbus ausgegeben. Anschließend wird der Zähler erhöht, er zeigt damit auf das nächste zu lesende Speicherwort. Ebenso wie beim Lesezähler wird der Schreibzähler nach Erreichen der höchsten Adresse beim nachfolgenden Lesezyklus zu Null gesetzt. Holt der Lesezähler den Schreibzähler ein, so sind alle Elemente der Schlange ausgelesen und der Speicher ist leer.



Zur Gewährleistung der korrekten Funktion darf weder der Lesezeiger den Schreibzeiger noch umgekehrt der Schreibzeiger den Lesezähler überholen. Der erste Fall stellt einen Unterlauf, der zweite Fall einen Überlauf dar. Im realen Einsatz ist zu spezifizieren, wie diese beiden Situationen behandelt werden. Häufig werden Kontrollsignale, welche die Über- und Unterlaufsituation signalisieren, nach außen geführt. Schlangenspeicher eignen sich sehr gut als Pufferspeicher für die Kommunikation zwischen zwei Prozessen. Der eine Prozeß schreibt die Schlange, der zweite liest sie. Der Speicher gewährleistet dabei, daß die Reihenfolge der Daten erhalten bleibt.

16.2.7 Assoziativspeicher (explizite Adressierung)

Beim Assoziativspeicher erfolgt die Auswahl eines Speicherworts explizit über einen Schlüssel. Dabei bilden die Schlüssel keinen kontinuierlichen Adreßraum, es muß lediglich die Unterscheidbarkeit unterschiedlicher Schlüssel gewährleistet werden, so daß ein Schlüssel eindeutig nur ein Speicherwort auswählt.



Der Speicher besitzt intern zwei Speicherfelder, ein Feld für die Daten und ein weiteres für die Schlüssel. Wird ein Schlüssel als Adresse an den Speicher angelegt, so wird er mit dem Inhalt jeder Schlüsselzelle verglichen. Ist der Schlüssel gültig, meldet genau ein Vergleicher die Gleichheit zur Schlüsselzelle und wählt damit das zugehörige Speicherwort aus.

Bei der realen Implementierung eines Assoziativspeichers sind noch die Fragen zu klären, welches Verhalten der Speicher bei einem ungültigen Schlüssel hat und wie ein Schlüssel in eine Schlüsselzelle eingetragen wird. Für das Verständnis der Funktionsweise ist dies nicht notwendig, daher werden diese Punkte an dieser Stelle nicht weiter ausgeführt.
Dieser Speichertyp findet sich mit fester Speicherwortlänge in schnellen Cache-Speichern.

17 Codierverfahren und Codesicherung

Für die im nachfolgenden Kapitel beschriebenen Massenspeicher sind Codier- und Codesicherungsverfahren von großer Bedeutung. Diese Verfahren wurden ursprünglich für die Kanalcodierung eingesetzt, die sich mit der Übertragung von Zeichen über gestörte Kanäle beschäftigte. Aber auch bei der Speicherung großer Datenmengen auf Magnetband oder Festplatte gibt es das Thema Störung der Daten. Deshalb wird zunächst auf die Codierverfahren eingegangen.

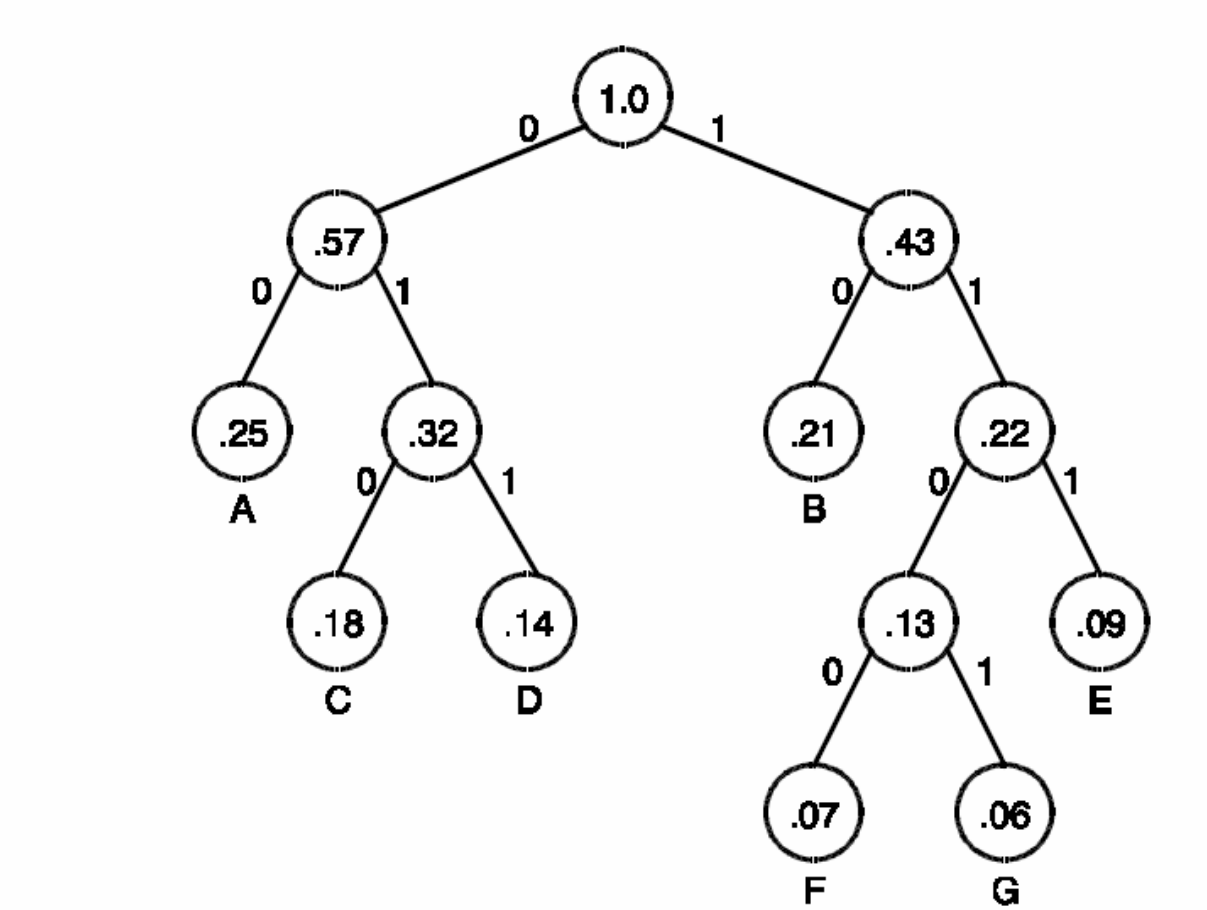
17.1 Huffman-Codierung

Je häufiger ein Zeichen z. B. in Texten auftritt - oder je wahrscheinlicher es ist - desto kürzer sollte sein Codewort sein. Für die Komprimierung wollen wir deshalb die ahrscheinlichkeiten p_i dafür, daß das i-te Zeichen an einer zufällig in der Zeichenkette ausgewählten Stelle auftritt, heranziehen und versuchen, die mittleren Codelänge (l_i ist die Länge des Codewortes für das i-te Zeichen)

$$L = \sum_{i=1}^{n} p_i l_i$$

zu minimieren. Dazu stellen wir den Huffman-Baum (H-Baum) auf.
Ein H-Baum ist ein Binärbaum. Er ist eine zweckmäßige Darstellung einer Informationsquelle und dient für die Auswahl, derjenigen Binärketten, die die Zeichen codieren sollen. Die Blätter des Baumes entsprechen den Zeichen (Nachrichten) der Informationsquelle und sind mit deren Wahrscheinlichkeiten beschriftet. Jeder weitere Knoten des H-Baums ist mit der Summe derjenigen Wahrscheinlichkeiten beschriftet, die den mit ihm verbundenen Knoten der nächst höheren Ebene (unter ihm) zugeordnet sind. Die beiden verbindenden Kanten werden mit 1 bzw. 0 beschriftet Es muß noch festgelegt werden, welche Knoten des Baums miteinander zu verbinden sind. Haben wir aber einen solchen H-Baum für die Informationsquelle erzeugt, so erhalten wir einen optimalen Code, wenn wir den Baum von der Wurzel zu den Blättern durchlaufen und uns die jeweilige Kantenbeschriftung merken.
Beispiel:
Eine Informationsquelle liefere die Zeichen A, B, C, D, E, F und G mit den Wahrscheinlichkeiten
0,25, 0,21, 0,18, 0,14, 0,09, 0,07 bzw. 0,06. Untenstehendes Bild zeigt den H-Baum dieser Informationsquelle.
Die Bitfolge, die eines der Zeichen codiert, ist die Folge von Bits, die man erhält, wenn man den Baum von der Wurzel aus zu diesem Zeichen verfolgt. Also im Beispiel:
A 00 B 10 C 010 D 011 E 111
F 1100 G 1101
Die durchschnittliche Codelänge L beträgt 2,67 Bits, und dies ist ein Minimum, wenn die Zeichen stochastisch unabhängig auftreten. (stochastische Informationsquelle). In der Regel sind Informationsquellen nicht stochastisch, d.h. die Wahrscheinlichkeit, daß z.B. ein bestimmter Buchstabe in einem Wort auf einen bestimmten vorliegenden Buchstaben folgt, ist

nicht unabhängig vom vorliegenden Buchstaben. Das folgende Verfahren läßt sich aber auch auf nichtstochastische Informationsquellen anwenden. Wie erhält man nun den H-Baum? Gegeben sei eine stochastische Informationsquelle mit den Elementarnachrichten (z.B. Zeichen) x_1, x_2, x_3, ..., x_n und den Auftrittswahrscheinlichkeiten p_1, p_2, p_3, ..., p_n. Den H-Baum erhält man wie folgt:



Ein H-Baum

Algorithmus von Huffman

1. Schritt: Suche zwei Zeichen x_i und x_j der Informationsquelle mit kleinsten Wahrscheinlichkeiten
2. Schritt: Bilde einen Knoten K_{ij} des Codebaums; ordne ihm die Wahrscheinlichkeit $p(K_{ij}) = p(x_i) + p(x_j)$ zu. Verbinde K_{ij} mit x_i und x_j.
3. Schritt: Entferne x_i und x_j aus der Informationsquelle und füge ihr statt dessen K_{ij} hinzu.
4. Schritt: Gehe zu Schritt 1, falls die Informationsquelle noch mehr als ein Zeichen enthält.
5. Schritt: Füge (letztes) Zeichen als Wurzel zum Codebaum. Beschrifte die Kanten wie oben erläutert.

Beispiel:

x_i	A	B	E	R
p_i	0,2	0,1	0,2	0,5

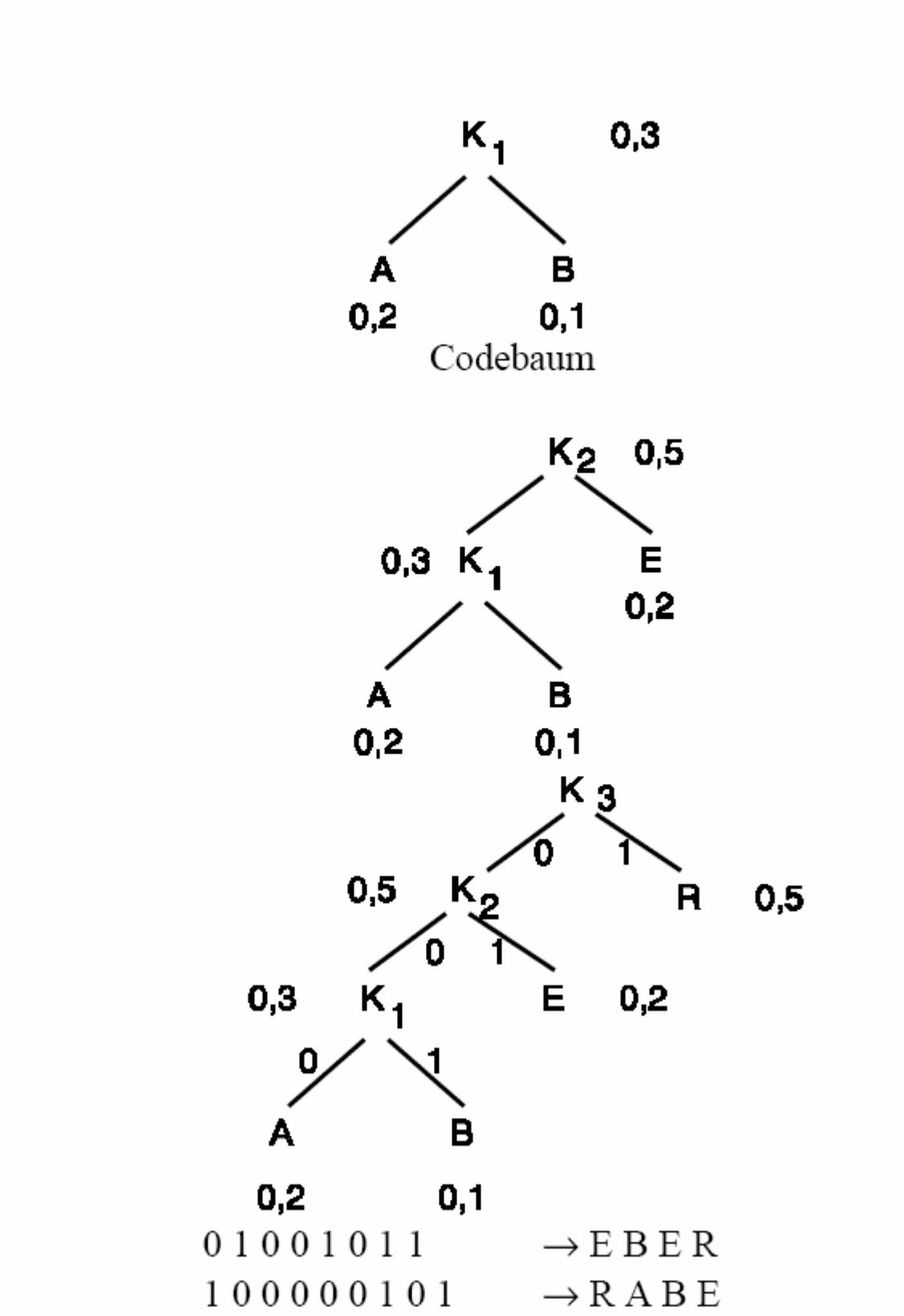
1. Schritt: $A, B \rightarrow K_1$

x_i	K_1, E, R
p_i	0,3 0,2 0,5

2. Schritt: $K_1, E \rightarrow K_2$

x_i	K_2, R
p_i	0,5 0,5

3. Schritt: $K_2, R \rightarrow K_3$



Die mittlere Codelänge für ein Zeichen beträgt somit:
$L = 3 * 0{,}2 + 3 * 0{,}1 + 2 * 0{,}2 + 1 * 0{,}5 = 1{,}8$ Bits.
Die optimale Codierung kann dann dem Codebaum entnommen werden. Der Code ist präfixfrei und optimal bezüglich der mittleren Codelänge. Die Präfixfreiheit ist wichtig, um einen Code einfach decodieren zu können, d.h. aus der Bitfolge das codierte Symbol bestimmen zu können.

17.2 Codesicherung, fehlertolerante Codierung

Nachrichten, die übertragen oder gespeichert werden, können verfälscht werden. Da bei der Codierung fast ausschließlich mit binären Zeichen (0,1) gearbeitet wird, kann es vorkommen, daß durch eine Störung oder ein Rauschen im Übertragungskanal ein Zeichen umgekehrt wird. Einigt man sich beispielsweise auf die Darstellung von 1 durch einen Spannungsimpulse von 4V und auf die Darstellung von 0 durch einen Impuls von 0V, so führt schon ein Störimpuls von 2V zu einem Fehler, da eine eindeutige Zuordnung dieser Spannung zu einem Binärzeichen nicht möglich ist. Um solchen Fehlern vorzubeugen, verwendet man Codes, die es dem Empfangsgerät ermöglichen, Fehler zu erkennen oder sogar zu korrigieren.
Bei einen ungesicherten Code werden alle Codeworte als Nutzworte (NW) gebraucht. Jede Störung führt zu einem neuen Nutzwort, welches vom Empfänger angenommen wird. Bei einem ungesicherten Code ist also keine Redundanz vorhanden.
Redundanz tritt auf, wenn bei einem Code mehr Informationen übertragen werden, als gebraucht werden. Je größer die Redundanz ist, desto besser ist die Fehlererkennung; diese verlangen aber auch höhere Kosten. Geht man z.B. von einem zweistelligen Code (EV = 2^2 = 4 Zeichen) A, B, C, D aus (A = 00, B = 01, C = 10, D = 11) und benutzt man alle Worte als Nutzworte, die Information übertragen können, so kann ein Fehler, z.B. bei A = 00 zu den neuen Worten B = 01 oder C = 10 führen. Dies bedeutet, daß keine Fehlererkennung möglich ist.
Die einfachste Art der Fehlererkennung und der möglichen Korrektur ist das Wiederholen einer Datenübertragung. Der Sender sendet die Information zweimal nacheinander und der Empfänger vergleicht. Verdoppelt die Wortlänge des Codes und erhöht stark die Redundanz. Eine weitere Möglichkeit ist das Handshake Verfahren, bei der der Sender wartet, dass der Empfänger das gleiche Wort zurückschickt. Bei einem Fehler wiederholt der Sender das Wort. Der Nachteil ist, dass bei beiden Verfahren einseitige oder unsymmetrische Störungen nicht erkannt werden. Die invertierte Wiederholung vermeidet diesen Nachteil dadurch, dass der Sender beim zweiten Mal alle Bits invertiert sendet. Eine einseitige Störung auf der Übertragungsleitung macht sich bei der Wiederholung umgekehrt bemerkbar und ist daher erkennbar. Der größte Nachteil der Nachrichtenwiederholung ist aber, dass doppelt so viele binäre Daten übertragen werden müssen als notwendig.

Um Fehler bei einer Übertragung zu vermeiden, darf bei einem Code nicht der gesamte Elementarvorrat als Nutzworte verbraucht werden. Einige Kombinationen müssen Pseudoworten (PW) zugeordnet werden. Diesen Pseudoworten wird zwar Binärcode aus dem Alphabet 2 zugeordnet, aber keine Information aus dem Alphabet 1. Diese Codes besitzen also eine Redundanz. Verwenden wir einen 3stelligen Code (EV = 2^3 = 8), so können beispielweise fünf Codeworte als Nutzworte (A,B, C, D, E) und drei Codeworte (F, G, H) als Pseudoworte verwendet.
Wir können also bei diesem fehlererkennenden Code - trotz des 3-stelligen Codes - nur 5 statt 8 Nachrichten übertragen.
Empfängt der Empfänger nun ein Pseudowort (110 oder 111), so erkennt er den Fehler und kann eine Wiederholung der Übertragung veranlassen.

Beispiel:
Eine weitere einfache Möglichkeit, Redundanz einzufügen besteht darin, für jede Null drei Nullen zu übertragen und eine Eins durch drei Einsen zu ersetzen. Angenommen, der Sender der Nachricht läuft synchron mit dem Empfänger, so daß der Beginn jeder Triade von 0 oder

1-sen bekannt ist, dann ist die Entschlüsselungsregel sehr einfach; sie ist in folgender Tabelle wiedergegeben.

Empfangener Wert	Bedeutung
000	0
100, 010 oder 001	0
011, 101 oder 110	1
111	1

Liegen wenigstens zwei gleiche Nachrichten vor, werden diese als gültig angesehen (Tripel Modular Redundancy (TMR) mit Votieren). Auf diese Weise erhält man die korrekte Nachricht auch dann, wenn in einem Exemplar ein Fehler auftritt.
Es sei nun p die Wahrscheinlichkeit, mit der ein einzelnes Bit verfälscht wird. Mit welcher Wahrscheinlichkeit wird dann eine Nachricht bei diesem Schema verfälscht? - mit der Wahrscheinlichkeit, daß mindestens 2 von 3 Bits verändert werden.
Wenn also 000 gesendet wird, kann daraus 110, 101, 011 oder 111 werden, und zwar mit den Wahrscheinlichkeiten
p^2q, pqp, qp^2 bzw. p^3, wobei q = 1-p ist.
Das Addieren dieser Wahrscheinlichkeiten ergibt $3p^2 - 2p^3$.
Dies bedeutet für einen Wert von p, der kleiner als 0,5 ist, daß das neue Schema eine wesentlich geringere Fehlerwahrscheinlichkeit hat.
Wenn z.B. p = 0,1 ist, berechnet sich die Wahrscheinlichkeit dafür, daß die Nachricht unwiederbringlich verfälscht wird, zu 0,028. Man hat also erreicht, daß trotz einzelner Fehler mit großer Wahrscheinlichkeit die richtige Information erhalten werden kann. Man sagt, die Codierung ist fehlertolerant - in diesem Fall genauer fehlermaskierend.

Nachricht	Bedeutung	Wahrscheinlichkeit: p Wahrscheinlichkeit für einen Bitfehler
000	0	(1 - p)(1- p)(1- p)
100	0	p(1- p)(1- p)
010	0	(1- p)p(1- p)
001	0	(1- p)(1- p)p
011	1	p(1- p)(1- p)
101	1	(1- p)p(1- p)
110	1	(1- p)(1- p)p
111	1	(1- p)(1- p)(1- p)

Umgekehrt betrachtet ist die Wahrscheinlichkeit, daß jeweils die richtige Nachricht erhalten wird:
$W = 3(1- p)^2 p + (1- p)^3$.

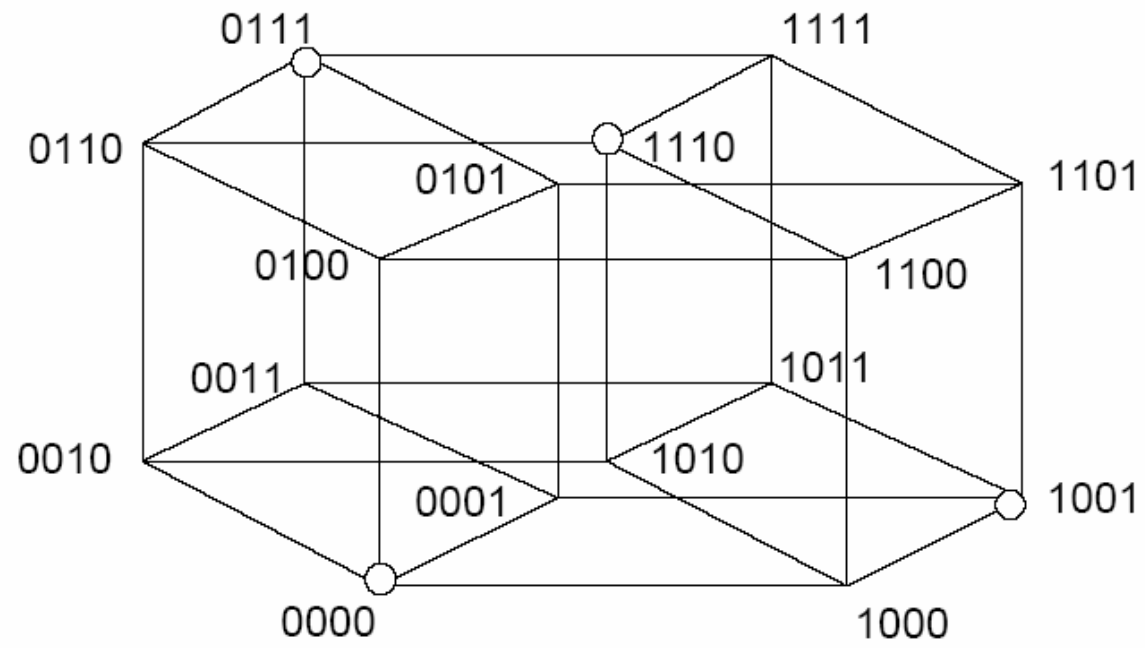
Das vorangegangene Codierungsschema benutzt einfache Redundanz als Schutz gegen Fehler. Es gibt aber auch andere, ausgefeiltere Verfahren. Man kann z.B. die Nachrichtenbits zu Paaren zusammenfassen und die Paare zusammen mit zwei zusätzlichen Checkbits (Prüfbits) entsprechend dem folgenden Schema übertragen.

Nachrichtenbits	Checkbits
a_1, a_2	a_3, a_4
0 0	0 0
0 1	1 1
1 0	0 1
1 1	1 0

Das erste Checkbit a_3 wiederholt einfach das zweite Nachrichtenbit a_2: $a_3=a_2$. Das zweite Checkbit wird Checksumme genannt. Es ist die "logische Summe" der ersten 2 Bits: $a_4=a_1$ xor a_2.

17.3 Stellendistanz und Hammingdistanz

Die Funktion dieser fehlerkorrigierenden Codierung wird an dem Hyperkubus deutlich.



Jede Ecke stellt eine mögliche 4-Bit-Kette dar. Vier der Ecken sind eingekreist. Sie entsprechen den vier ursprünglichen (unverfälschten) Codewörtern. Ecken, die 4-Bit-Ketten darstellen, die sich in nur einem Bit unterscheiden, sind durch eine Kante verbunden. Von den 16 4- Bit-Ketten sind 12 keine Codewörter. Sie könnten aber durch Bitfehler aus einem Codewort entstanden sein. Wenn man jedes dieser Wörter untersucht, stellt man fest, daß man bei einer kleinen Fehlerrate am besten jedes Wort als das Codewort neu interpretiert, das mit ihm durch den kürzesten Kantenzug verbunden ist.
Der hier benutzte Abstandsbegriff ist die **Stellendistanz** zweier Bitfolgen. Er entspricht der Anzahl unterschiedlicher Bits in diesen Bitfolgen.

Stellendistanz ist die Anzahl von Stellen, in denen sich zwei gültige Codeworte eines Codes voneinander unterscheiden.

Beispiel

Betrachten wir jetzt ein 3-Bit Dual-Code, in welchen 4 Worte der Informationsübertragung dienen und 4 weitere ohne irgendeine Informationszuweisung sind – diese sind also Pseudoworte.

A 000
-- 001
B 010
-- 011
-- 100
C 101
D 110
-- 111

Die Stellendistanz d beträgt hier jeweils:
Zwischen A & B = 1
Zwischen B & C = 3
Zwischen C & D = 2
Zwischen A & D = 2

Der Hamming-Abstand einer Codierung ist das Minimum der Stellendistanzen aller Codewort-Paare.

Die Hammingdistanz h ist die kleinste Anzahl von Stellen, in denen sich zwei gültige Codeworte eines Codes voneinander unterscheiden (DIN 44 300) - es ist also die kleinste Stellendistanz im Code.

$$h = \min d$$

Im Beispiel ist die Hammingdistanz gleich 1.
Wenn die Codewörter einen genügend großen Abstand haben, also der Hamming-Abstand der Codierung genügend groß ist, wird man so tolerant gegen mögliche Einzelfehler. Zu viele Fehler führen allerdings unweigerlich zu einem falschen Ergebnis. Es gilt, den Hamming-Abstand so groß wie nur möglich zu machen.
Im obigen Beispiel ist der Hamming-Abstand allerdings nur gleich 2. D.h. nicht jeder Fehler wird richtig korrigiert, wenn man den „nächsten Nachbarn“, der einem Codewort entspricht, wählt. Deshalb kann schon ein einzelner Fehler zu einem falschen Korrekturergebnis führen. Ein einzelner Fehler wird aber immer erkannt.

Beispiel

Betrachtet man nun einen (2-aus-4)-Code, beim dem es 6 Möglichkeiten gibt. Allerdings werden zur Informationsübertragung in unserem Fall nur 4 benutzt. Alle Zeichen treten gleichhäufig auf.

A 0011
  0101
B 0110
C 1001
  1010
D 1100

Die Stellendistanz **d** beträgt hier jeweils:
Zwischen A & B = 2
Zwischen B & C = 4
Zwischen C & D = 2
Zwischen A & C = 2
Zwischen B & D = 2
Zwischen A & D = 4

Deshalb ist die Hammingdistanz Minimum von allen d Werten
h = 2 [Bit/Zeichen].

Es ist ein gleichlanger Code. Die Wortlänge beträgt hier →
L = 4 [Bit/Zeichen],

Die Entropie $H = \Sigma\, p_i ld(1/p_i) = ld\ 4 = 2$. Dadurch beträgt die Redundanz
R = L – H = 2 [Bit/Zeichen]

Wie könnte man es erreichen, dass die Hamming Distanz größer wird?
Man kann zum Beispiel einen 5-stelligen Code benutzen.

Beispiel
Betrachten wir jetzt einen 5-stelligen Code, aus welchem nur 4 Elemente als Nutzworte ausgewählt sind, bei Annahme der Gleichhäufigkeit:
A 00000
B 01011
C 10101
D 11110

Die Stellendistanz **d** beträgt hier jeweils:
Zwischen A & B = 3
Zwischen B & C = 4
Zwischen C & D = 3
Zwischen A & C = 3
Zwischen B & D = 3
Zwischen A & D = 4

Die Hammingdistanz h ist grösser geworden
h = min d = 3.

Hier wäre die Wortlänge
L = 5,

Entropie H = ld 4 = 2

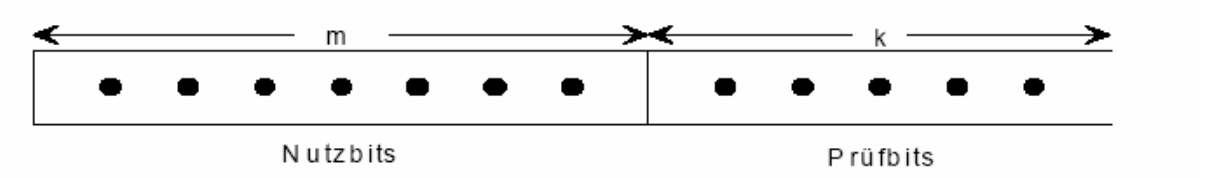
und dadurch die Redundanz
R = L–H = 3.

Also ist die Hammingdistanz ist grösser geworden und die Redundanz auch.

Durch zusätzliche Bits in der Codierung kann man also eine Fehlererkennung und auch eine Fehlerkorrektur erreichen. Für das Übertragen von Nachrichten, die nur fehlererkennend, aber nicht fehlerkorrigierend codiert sind, ist erforderlich, daß der Empfänger den Sender zur Wiederholung veranlassen kann, falls bei der Übertragung ein Fehler entstand und dies vom Empfänger erkannt wurde. Bei fehlerkorrigierenden Codes ist dies nicht nötig, sie sind allerdings technisch aufwendiger zu realisieren. Eine einfache Fehlererkennung wird durch die Verwendung eines einzigen Prüfbits (Paritätsbits) möglich. Jedem Wort wird ein weiteres Bit hinzugefügt, so daß die Summe aller 1-Bits für jedes Worte immer gerade (oder immer ungerade) ist (Hamming-Abstand 2).

17.4 Fehlererkennung, Fehlerkorrektur und Coderedundanz

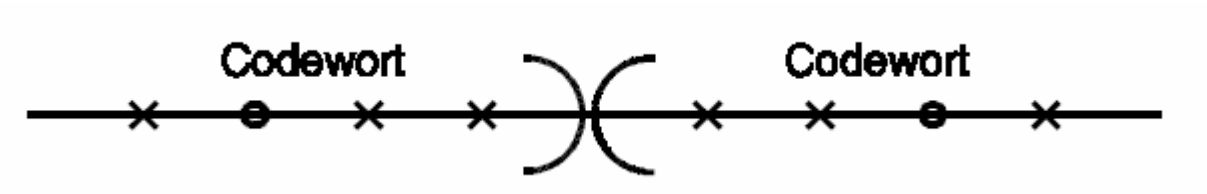
Fehlertoleranz basiert, wie wir gesehen haben, darauf, daß der verwendete Code nicht optimal ist, sondern (nützliche) Redundanz enthält, was bedeutet, daß der Vorrat an möglichen Codeworten nicht vollständig ausgeschöpft wird. Die benutzten Codewörter bezeichnet man als Nutzwörter, diejenigen, die nicht Codewort sind, als Pseudowörter. Die redundanten (zusätzlichen) Bits nennt man Prüfbits.
Mit s zusätzlichen Bits:
Wieviele Fehler können erkannt werden?
Wieviele Fehler können korrigiert werden?
Die Antwort hängt von der Coderedundanz und dem Berechnungsverfahren für die Prüfbits ab. Unter der Coderedundanz wird das Verhältnis r = k/m verstanden, wobei n die Länge des Codeworts, m die Anzahl der Nutzbits und k = n-m die Anzahl der zusätzlichen Prüfbits ist.



Es gilt der Satz:
Um alle möglichen 1-Bit-Fehler in einem Bitmuster der Länge m korrigieren zu können (damit der Hamming-Abstand mindestens gleich 3 wird), sind mindestens k zusätzliche Prüfbits erforderlich, wobei
$\mathbf{n = m + k < 2^k}$
gelten muß.
Z.B. erfüllen m = 4 und k = 3 diese Bedingung, da dann $m + k = 7 < 2^3 = 8$ ist. Außerdem ist r = (7-4)/4 = 0,75.
Hierzu zählt auch das Tripletten-Schema (s.o) mit: m = 1, k = 2.
Weitere Kombinationen sind in untenstehender Tabelle angegeben.

m	4	8	11	16	26
k	3	4	4	5	5
r	0,75	0,50	0,36	0,32	0,19

Der Beweis hierzu folgt ein paar Seiten später.
Ein Hamming-Abstand d erlaubt die Erkennung von d - 1 Bitfehlern und eine Korrektur von (d - 1)/2
Fehlern, wenn d ungerade ist, und von
d/2 - 1
Fehlern, wenn d gerade ist.



Hamming-Abstand d = 5

Zum Bild: Falls höchstens 2 Bit verfälscht sind, ist eine Korrektur möglich. Es können aber bis zu 4 Bitfehler erkannt werden. Wenn aber bis zu 4 Bits fehlerhaft sein können, ist eine sichere Korrektur nicht mehr möglich - auch nicht eines einzelnen Bitfehlers. D.h. man muß sorgfältig unterscheiden, ob man mögliche Fehler nur erkennen oder aber auch korrigieren will. Fehlererkennung und Fehlerkorrektur können auch miteinander kombiniert werden. Bei einem SEC/DED-Code (single error correcting, double error detecting code) läßt sich jeder Einzelbit-Fehler korrigieren. Treten aber zwei Bitfehler auf, dann läßt sich dies erkennen und auch feststellen, daß zwei Bitfehler vorliegen und somit eine sichere Korrektur nicht möglich ist. Folglich darf dann eine Korrektur auch nicht versucht werden.

Aufgabe:
Man zeige, daß der folgende Code ein SEC/DED-Code ist.

k = 4, n = 8 (Die Verknüpfungsoperation der Bits ist das exklusive Oder, s.u.)
Bitposition / P_i Prüfbit, D_i Nutzbit

8	7	6	5	4	3	2	1
P_4	D_3	D_2	D_1	P_3	D_0	P_2	P_1

$P_1 = D_3 \oplus D_1 \oplus D_0$, $P_2 = D_3 \oplus D_2 \oplus D_0$, $P_3 = D_3 \oplus D_2 \oplus D_1$, $P_4 = D_2 \oplus D_1 \oplus D_0$

Ein Rechner ist fehlertolerant, wenn er seine Fehler selbsttätig erkennen und korrigieren kann. Diese Eigenschaft wird in zuverlässigkeits-kritschen Anwendungen verlangt. Natürlich lassen sich nie alle der möglichen Fehler erkennen und korrigieren. Fehlererkennung- und korrektur ist deshalb nur mit einer gewissen Wahrscheinlichkeit möglich - mit der Wahrscheinlichkeit, dass ein erkennbarer bzw. korrigierbarer Fehler aufgetreten ist.

Was ist der praktische Unterschied zwischen einer Fehlererkennung und einer Fehlerkorrektur? Das kann man an folgendem Beispiel zeigen.

Beispiel
Betrachten wir einen (3-aus-5)-Code. Dieser Code enthält insgesamt 10 Nutzwörter, weil

$$N = \frac{5!}{3!(5-3)!} = \frac{5*4}{2} = 10$$

In jedem Codewort müssen jeweils 3 Stellen "1" und 2 Stellen "0" aufweisen.

Zahl	Code	d	
0	00111	2	
1	01011		2
2	01101	2	
3	01110		4
4	10011	2	
5	10101		2
6	10110	4	
7	11001		2
8	11010	2	
9	11100		4

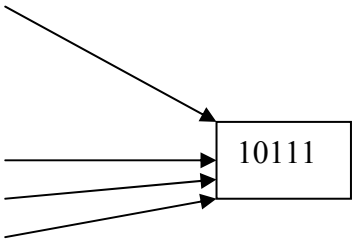


Abb. Fehlererkennung

Die Stellendistanz **d** ist hier verschieden - in der Abbildung sind nur die ersten Paare von Worten verglichen, man müsste es weiter machen - aber d variiert zwischen 2 und 4.
Die Hammingdistanz ist also h=2. Deshalb können 1-Bit Fehler erkannt werden. Aber welcher Fehler es ist, kann man nicht sagen. Wenn das verfälschte Datenwort z.B. 10111 lautet, dann es kann mit 1 verfälschten Bit aus 4 Nutzworten entstanden sein (siehe Abb.Fehlererkennung).
Die Erkennung dieses Fehlers ist möglich (e=h-1) aber eine Selbstkorrektur noch nicht.

Tritt bei der dargestellten Zuordnung ein Übertragungsfehler auf, so wird ein Pseudowort empfangen, welches wieder dem ursprünglichen Nutzwort zugeordnet werden kann.
Nun kann man die zu erkennenden und zu korrigierenden Fehler berechnen.
Wie wird es aber realisiert? Es geht nach dem Hamming Verfahren. Die Datenbits werden mit zusätzlichen Prüfbits ergänzt, nach den von Hamming gegebenen Regeln - diese Prüfbits bilden mit bestimmten Datenbits die gerade Quersumme. Das Verfahren geht davon aus, daß das Auftreten eines einzelnen 1-Bit Übertragungsfehlers in einem Codewiort wahrscheinlicher ist, als daß während der Übertragungsdauer eines Codewortes mehrere Bitfehler entstehen.
Die Gesamtzahl B der Bits eines übertragenen Codewortes nach dem Hamming Verfahren setzt sich aus einer Anzahl von Datenbits D und einer Anzahl von Prüfbits P zusammen:

$$B = D + P$$

Um nun als Prüfungsergebnis die Nummer der fehlerhaften Bitstelle des übertragenen Codewortes zu erhalten, ordnet man den Kombinationen der Prüfbits entsprechende Bitnummern zu. Daher gilt als Voraussetzung, dass die mit einer Anzahl P von Prüfbits maximal darstellbare Bitnummer grösser oder zumindest gleich der Gesamtsumme der Datenbits D und der Prüfbits P sein muss. Da sich mit der Anzahl P von Prüfbits allgemein $2^P - 1$ Bitstellen kennzeichnen lassen, gilt daher die folgende Ungleichung, an hand der sich die Anzahl der benötigten Prüfbits P berechnen lässt:

$$2^P - 1 \geq B$$

Für beispielweise 4 Datenbit-Worte benötigt man dann 3 Prüfbits, weil

Bei P = 1 $\Rightarrow$ $2^1 - 1 = 1$ 4+1=5 nein
Bei P = 2 $\Rightarrow$ $2^2 - 1 = 3$ 4+2=6 nein
Bei P = 3 $\Rightarrow$ $2^3 - 1 = 7$ 4+3=7 ja !

Die Reihenfolge der Übertragung der einzelnen Daten- und Prüfbits wird jetzt so gewählt, daß das Prüfergebnis (als Dualzahl betrachtet) der Nummer des fehlerhaften Bits entspricht. Dazu müssen die Prüfbits die Stellen der Zweierpotenzen des neuen Codeworts einnehmen:

$P_0 \Rightarrow 2^0$ -, $P_1 \Rightarrow 2^1 = 2$, $P_2 \Rightarrow 2^2 = 4$.

Das Codewort wird folgende Struktur haben: **D_3 D_2 D_1 P_2 D_0 P_1 P_0 .**
Die gerade Quersumme von ausgewählten Datenbits wird mit folgenden Gleichungen bestimmt:

$P_0 = D_0 \square D_1 \square D_3$
$P_1 = D_0 \square D_2 \square D_3$
$P_2 = D_1 \square D_2 \square D_3$

Die Datenbits D_0, D_1, D_2, D_3 werden um Prüfbits P_0, P_1, P_2 ergänzt, nach den oben gegebenen Regeln. Dann es wird aus 4-Bit-Code ein 7-Bit Code Hamming-Code wie unten. In der Tabelle sind nur 10 Worte gezeigt, obwohl das Verfahren für alle 16 Kombinationen von 4 Datenbits gilt.

Info	D_3	D_2	D_1	P_2	D_0	P_1	P_0
0	0	0	0	**0**	0	**0**	**0**
1	0	0	0	**0**	1	**1**	**1**
2	0	0	1	**1**	0	**0**	**1**
3	0	0	1	**1**	1	**1**	**0**
4	0	1	0	**1**	0	**1**	**0**
5	0	1	0	**1**	1	**0**	**1**
6	0	1	1	**0**	0	**1**	**1**
7	0	1	1	**0**	1	**0**	**0**
8	1	0	0	**1**	0	**1**	**1**
9	1	0	0	**1**	1	**0**	**0**

Wenn die Elemente gleichhäufig auftreten, dann gilt
$H = H_0 = ld\ 10 = 3.32$ [Bit/Zeichen],
$L = 7$ [Bit/Zeichen],
die Redundanz $R = L-H = 3.68$ [Bit/Zeichen].
Dabei ist die relative Redundanz gross $r = 3.68/7 = 52.5\%$.
Die Hammingdistanz ist $h = 3$ [Bit/Zeichen].
Dann werden die so erweiterten Zeichen mit Information gesendet, wobei die Positionen der Datenbits und Prüfbits festgelegt sind. Der Empfänger errechnet aus den empfangenen Datenbits die Prüfbits und vergleicht sie mit den Prüfbits, die er ampfangen hat. Die Differenz zwischen errechneten und empfangenen Prüfbits zeigt, welche Stelle fehlerhaft ist.

Fehlerbit Tabelle

Fehlerwert			Fehler	in
FP2	FP1	FP0	Bit-Nr.	Bit
0	0	0	-	-
0	0	1	1	P0
0	1	0	2	P1
0	1	1	3	D0
1	0	0	4	P2
1	0	1	5	D1
1	1	0	6	D2
1	1	1	7	D3

Beispiel

Es wird eine Information im BCD-Code gesendet.
Es ist ein gleichlanger Code mit der Länge
L=4 Bit.
Die Datenbits wurden um 3 Prüfbits nach dem Hammingverfahren ergänzt.
Der Empfänger hat das Wort **0010001** empfangen. Dieses Wort gehört nicht zu den Nutzworten unsere Alphabets. Welches Bit ist fehlerhaft?
Lösung:
Der Empfänger errechnet die Prüfbits aus allen empfangenen Bits.

$RP_0 = ED_0 + ED_1 + ED_3$
$RP_1 = ED_0 + ED_2 + ED_3$
$RP_2 = ED_1 + ED_2 + ED_3$

wobei RP die Rechenprüfbits sind (Prüfbits errechnet aus den empfangenen Datenbits ED).
In diesem Fall ergibt sich:

$RP_0 = 0 + 1 + 0 = 1$
$RP_1 = 0 + 0 + 0 = 0$
$RP_2 = 1 + 0 + 0 = 1$

Die Prüfbitsfolge 101 unterscheidet sich von der empfangenen 001. Das Fehlerwort errechnet man als Differenz aus beiden: 100.
Mit diesem Wert gehen wir in die Fehlertabelle und wissen schon, dass das fehlerhafte Bit P_2 ist. Das könnten wir auch ohne Tabelle wissen, da 100 der dezimalen 4 entspricht. Dies bedeutet, daß das fehlerhafte Bit an der Stelle 4 (von rechts gezählt) ist - und an der Stelle Nummer 4 ist Bit P_2. Jetzt wird der Fehler korrigiert - das fehlerhafte Bit wird invertiert – und das richtige Wort sollte **0011001** sein, demnach das Datenwort ohne Prüfbits **D=0010**.

Beispiel

Mit dem selben Code, wie im vorigen Beispiel wird jetzt das Wort **1000101** empfangen. Offensichtlich gehört es nicht zu den Nutzworten, die in der Tabelle zusammengefasst sind. Es ist also fehlerhaft. Wo ist der Fehler?
Lösung:

Errechnetes Prüfwort:

$RP_0 = 1 + 0 + 1 = 0$
$RP_1 = 1 + 0 + 1 = 0$
$RP_2 = 1 + 0 + 0 = 1$

RP=100 unterscheidet sich vom empfangenen Prüfwort EP=001. Die logische Differenz ergibt 101, also Position 5, ds entspricht D1. Nach der Korrektur heiß das neue korrigierte empfangene Datenwort **1010101**. Das neu berechnete Prüfwort ergibt sich zu RP=001. Das empfangen Prüfwort ist unverändert, die Differenz ist also 000.
Das würde eigentlich bedeuten, daß kein Fehler aufgetreten ist. Da es aber kein Wort aus unserem Alphabet ist, bedeutet es, daß in dem empfangenen Wort mehr als ein Fehler aufgetreten ist, und dafür ist das Verfahren - mit dieser Anzahl Prüfbits - nicht geeignet.

Die Anzahl der mit Sicherheit zu erkennenden Fehler ist
e = h – 1

h = 1	Code ist weder prüfbar noch korrigierbar
h ≥ 2	Code ist prüfbar auf (h–1) Bits
h ≥ 3	Code ist korrigierbar auf (h-1)/2 Fehlerbits, wenn h ungerade

oder auf (h/2)-1 Fehlerbits, wenn h gerade.

17.5 Parity-Bit

Bei der Daten-Übertragung wird das achte oder neunte Bit ("parity bit") oft zur Fehlererkennung ("parity check") verwendet.
Dieses "Parity-Bit" wird vor der Übertragung meist so gesetzt, daß die Summe aller Nutzbits (also Bits, die Information tragen) geradzahlig (= 0) wird. Beim Empfang wird diese Summe überprüft und andernfalls nochmals angefordert. Das ist Grundlage des Verfahrens "Fehlererkennung durch Paritätsprüfung" (parity check).
Ungesicherte Codes können durch die Hinzunahme eines Prüfbits (parity bit) aufgestuft werden - 1-Bit-Fehler werden erkannt. Zwei Codeworte, die sich zunächst nur um eine Stelle unterscheiden, unterscheiden sich mit dem Paritätsbit nun um zwei Stellen. Der Wert des Paritätsbits wird folgendermaßen ermittelt:

- gerade Parität (even parity): Die Anzahl der 1-Bits im erweiterten Codewort ist gerade. Das Wort wird auf eine gerade Anzahl von 1-Bits erweitert.
- ungerade Parität (odd parity): Die Anzahl der 1-Bits im erweiterten Codewort ist ungerade. Das Wort wird auf eine ungerade Anzahl von 1-Bits erweitert.

Die Erweiterung des Codewortes um ein Paritätsbit wird auch "Querparität" oder "Zeichenparität" genannt. Die Überprüfung der empfangenen Codeworte kann durch Bildung der Quersumme (Addition modulo 2, d.h. ohne Berücksichtigung des Übertrags) erfolgen.

Das Paritätsbit ergänzt diese Anzahl auf einen geraden oder ungeraden Wert, je nach Verfahren. Bei Verwendung der Paritätsprüfung wird eine ungerade Zahl von Fehlern in einem Block erkannt damit auch alle einzelnen Fehler. Gerade Fehlerzahlen heben sich gegenseitig in ihrer Wirkung auf das Paritätsbit auf und können mit diesem Verfahren nicht entdeckt werden.

17.6 Blocksicherung

Blocksicherung oder Blockprüfung ist ein Verfahren, in welchem die Daten für die Übertragung in einen Block zusammengefasst sind.

Codewort	D3	D2	D1	D0	PB
CW1	0	1	0	1	**1**
CW2	1	1	0	0	**1**
CW3	1	0	0	0	**0**
CW4	1	1	**1**	1	**0**
CW5	0	0	0	0	**1**
Prüfwort	**0**	**0**	**1**	**1**	**0**

Abbildung Blocksicherung.

Jedem Codewort des Datenblockes wird ein Parity-Bit (PAR) angehängt, entsprechend der Vereinbarung ein gerades oder ungerades Paritäts-Bit. In der Abbildung wurde ungerade Parität gewäehlt, d.h. die Summe der "1" in der Zeile zusammen mit dem Prüfbit soll ungerade sein. Zusätzlich wird der Datenblock durch ein Prüfwort gesichert, welches nach der Übertragung des letzten Codewortes dem Datenblock angehängt wird. Das Prüfwort besteht also aus Prüfbits für die jeweilige Spalte. Die Spalte von Parity-Bits und die Zeile des Prüfwortes werden ebenfalls mit einem Parity-Bit gesichert. In obiger Abbildung haben wir ein Block von 5 Codewörtern, wovon 1 Block fehlerhaft übertragen wurde.
Der Empfänger überprüft nach dem Empfang jedes einzelnen Codewortes anhand des angehängtes Parity-Bits auf seine Fehlerfreiheit. Im Falle nur eines Fehlers in einem der 5 Codewörter kann man anhand des Prüfwortes die Bitstelle des fehlerhaften Datenbits ermitteln und korrigieren.
Je geringer die Anzahl der Codewörter pro Datenblock gewählt wird, um so weniger wahrscheinlich wird das Auftreten mehrerer Fehler während der Übertragung eines Datenblockes. Ein Übertragungsfehler kann entweder in den Codewörtern oder im Prüfwort auftreten. In jedem Fall kann aber ein 1-Bit-Fehler erkannt und korrigiert werden.
In obiger Abbildung war das Datenbit D1 des Codewortes CW4 verfälscht - das korrekte Codewort CW4 lautet 1101.
Treten dagegen während der Übertragung eines Datenblockes zwei oder mehr als zwei Übertragungsfehler innerhalb eines Datenblockes - der die vereinbarte Anzahl von Codewörtern und das Prüfwort umfasst - auf, so lässt sich eine Korrektur nicht mehr durchführen, obwohl der Fehler noch signalisiert wird. Dies könnte z.B. auftreten, wenn außer CW4 auch noch in CW3 das Datenbit D1 falsch übertragen wurde.

Codewort	D3	D2	D1	D0	PB
CW1	0	1	0	1	**1**
CW2	1	1	0	0	**1**
CW3	1	0	**1**	0	**0**
CW4	1	1	**1**	1	**0**
CW5	0	0	0	0	**1**
Prüfwort	**0**	**0**	**1**	**1**	**0**

Abbildung: Doppelfehler bei Blocksicherung.

Hier kann zwar der Empfänger die Fehler in den Zeilen CW3 und CW4 feststellen, aber er kann nicht die fehlerhafte Bits bestimmen - das Prüfwort erscheint wegen des Doppelfehlers als richtig!
Blocksicherungverfahren finden ihre praktische Anwendung z.B. bei der Datenspeicherung auf magnetischen Massenspeichern.

Beispiel
Betrachten wir ein Magnetband, welches mit 5 Spuren beschrieben wird. Die Spuren beinhalten Codewörter, ergänzt um ein Prüfbit. In einem Block haben wir also 4 Codewörter, jedes Wort besteht aus 6 Datenbits und einen Prüfbit, welches der Magnetbandcontroller berechnet und anhängt. Zwischen einzelnen Blocks sind kleine leere Abstände, in denen nichts gespeichert wird (engl. Gap). Die Prüfbits sind für jeden Block in einer Spalte hinter den Datenbits zusammengefasst. Diese Spalte nennt man LRC (Longitudinal Redundancy Check = Waagerechte Redundanz Prüfung). Einzelne Datenbits aus allen Codewörtern werden zusätzlich aufaddiert und in der letzten Spur in einem Prüfwort zusammengefasst. Dieses Prüfwort nennt man hier VRC (Vertical Redundancy Check = Vertikale Redundanz Prüfung). Bei einem späteren Lesevorgang errechnet der Controller erneut die Prüfbits aus

den Datenbits und vergleicht sie mit den ursprünglich gespeicherten Prüfbits. Mögliche Fehler werden korrigiert. Diese Verfahren nennt man "Read after Write" (= Lesen nach dem Schreiben). In untenstehender Abbildung sind keine fehlerhafte Daten- oder Prüfbits enthalten.

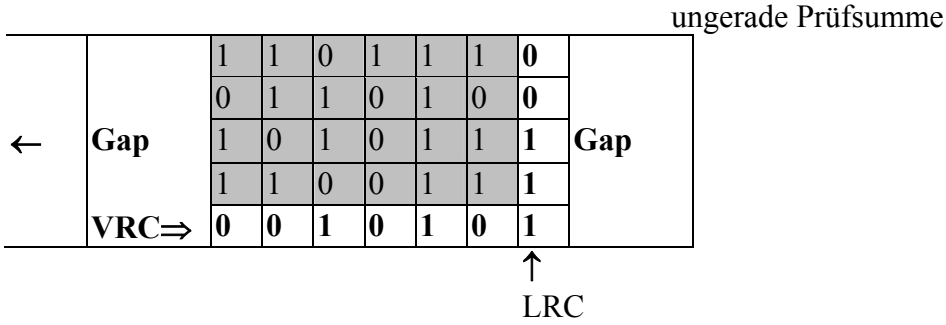


Abbildung: Magnetband mit Blocksicherung.

17.7 Zyklischer Redundanzcode (CRC-Cyclic Redundancy Code, Polynomcode)

Mit einer CRC-Prüfsumme können Fehler detektiert aber nicht korrigiert werden. Daher ist ein CRC-Verfahren immer dann sinnvoll, wenn bei der Detektion eines Fehlers die fehlerbehaftete Bitfolge erneut übertragen werden kann.
Alternativ werden CRC-Verfahren mit Querparitäten kombiniert, so daß dann doch Fehler auch korrigiert werden können.
Das CRC-Verfahren wird in der Praxis sehr häufig eingesetzt da es eine gute Fehlererkennung gewährleistet, nur wenig Aufwand verursacht und sehr einfach in Hardware zu realisieren ist.

Von einem Sender soll an einen Empfänger eine Nachricht (N) gesendet werden, die aus einer Bitfolge besteht. Der Empfänger soll eine Möglichkeit erhalten zu überprüfen ob auf dem Übertragungsweg die Bitfolge verändert bzw. gestört wurde. Hierzu wird für jede Bitfolge eine Prüfsumme (P) berechnet, indem eine mathematische Operation auf der Bitfolge ausgeführt wird. Die Prüfsumme wird mit der Nachricht zu N* zusammengefasst und versendet. Der Empfänger kann mit Hilfe der Prüfsumme und der ihm bekannten Vorschrift zur Berechnung die Konsistenz der übertragenen Nachricht überprüfen. Der Begriff Prüfsumme hat sich zwar durchgesetzt, bezeichnet im Rahmen des CRC-Verfahrens aber nicht eine Addition der Daten der Bitfolge. Das Prinzip des CRC-Verfahrens basiert auf der Division der Bitfolge der Nachricht mit einem standardisierten Divisor. Der sich bei der Division ergebende Rest wird so mit der Nachricht zu N* verknüpft, dass der Rest bei einer Division der resultierende Bitfolge mit dem standardisierten Divisor verschwindet. Der Empfänger kann nun die empfangene Bitfolge mit dem ihm bekannten Divisor teilen und so die korrekte Übertragung überprüfen. Im folgenden wird diskutiert wie die Division so definiert werden kann, dass die Berechnung effizient realisiert werden kann. Darüber hinaus wird kurz skizziert welche Aspekte bei der Standardisierung der Divisoren zu berücksichtigen sind, so dass eine optimierte Fehlererkennung gewährleistet ist.

Die Division wird nach der algebraischen Feldtheorie Modulo 2 definiert. Hierzu wird jede Bitfolge als Polynom dargestellt. Aus der beispielhaften binären Bitfolge 1001 wird

$1*x^3+0*x^2+0*x^1+1*x^0$, dies kann zu x^3+1 vereinfacht werden. Es handelt sich hierbei um ein Polynom der Ordnung 3. Im Rahmen der hier verwendeten mathematischen Operationen mit Polynomen wird von der konkreten Wertigkeit der Koeffizienten abstrahiert. D.h. insbesondere, dass die Gleichheit der Basis der einzelnen Glieder nicht gewährleistet ist. Aus diesem Grund macht es wenig Sinn bei der Definition von Rechenoperationen Überträge zu berücksichtigen. Wenn als Rechenvorschrift Modulo 2 verabredet wird, ergeben sich folgende Additions- und Subtraktionsregeln

1. Bit	2. Bit	Addition	Subtraktion
0	0	0	0
1	0	1	1
0	1	1	1
1	1	0	0

Eine Analyse der Tabelle zeigt, dass sich die Rechenregeln für Addition und Subtraktion gleichen und mit der XOR-Funktion beschrieben werden können. Vor dem Hintergrund, dass keine Überträge berücksichtigt werden, unterscheiden sich Addition und Subtraktion auch für längere Bitfolgen nicht. Aus diesem Grund kann im Rahmen der algebraischen Feldtheorie Modulo 2 für Bitfolgen gleicher Länge keine Größenrelation angegeben werden. Eine Division wird nach den gleichen Regeln berechnet wie für den Fall von Dezimalzahlen, d.h. die Division ist eine sukzessive Subtraktion. Zur Durchführung der Division für das CRC-Verfahren muss der Divisor bekannt sein, der in diesem Zusammenhang auch als Generatorpolynom G bezeichnet wird. Die Ordnung des Generatorpolynoms wird mit g bezeichnet. Zur Durchführung der Division werden an die niederwertige Seite der Bitfolge der Nachricht N g Nullbits angehängt. Durch diese Operation entsteht N*. Anschließend wird N* durch G geteilt. Der Quotient ist für die weitere Verarbeitung ohne Interesse. Der berechnete Rest der Division wird von N* subtrahiert. Auch vor dem Hintergrund der Division von Dezimalzahlen ist nun offensichtlich, dass N* ohne Rest durch G teilbar ist. N* wird an den Empfänger übertragen. Der Empfänger überprüft ob die von ihm empfangene Bitfolge ebenfalls ohne Rest durch das ihm bekannte Generatorpolynom geteilt werden kann. Falls der Rest verschwindet, kann der Empfänger die vom Sender angefügten g Bit entfernen und auf diese Weise N erhalten. Im anderen Fall muss die Übertragung der Bitfolge wiederholt werden. N* wird dann fälschlicherweise als korrekt angenommen, wenn das gestörte N* ebenfalls ein Vielfaches des Generatorpolynoms ist. Es gilt nun das Generatorpolynom so zu wählen, dass möglichst viele Fehlermöglichkeiten ausgeschlossen werden können.

Bei der Übertragung von N* können einzelne Bitfehler auftreten, d.h. ein einziges Bit der Bitfolge kann kippen. Derartige Fehler können ausgeschlossen werden indem das Generatorpolynom mindestens 2 Koeffizienten enthält die 1 sind. Neben Einzelfehlern können auch Zwei-Bit-Fehler auftreten. Auch dieser Fall kann durch eine entsprechende Konstruktion des Generatorpolynoms zuverlässig detektiert werden. Für eine höhere Anzahl von Bitfehlern kann entweder der Fall einer ungeraden Anzahl von Fehlern oder der Fall einer geraden Anzahl von Bitfehlern durch entsprechende Konstruktion vollständig ausgeschlossen werden. Sogenannte Burstfehler äußern sich in einer zusammenhängenden Folge von Einsen innerhalb von N*. Dieser Fehlerfall kann durch die Wahl von 1 für das niederwertigste Bit des Generatorpolynoms zuverlässig detektiert werden. Auf Basis dieser Konstruktionsanforderungen wurden einige Generatorpolynome für unterschiedliche Anwendungen standardisiert:

Bezeichnung	Polynom	Anwendung
CRC-12	$x^{12}+x^{11}+x^3+x^2+x^1+1$	6 Bit Werte
CRC-16	$x^{16}+x^{15}+x^2+1$	8 Bit Werte
CRC-CCITT	$x^{16}+x^{12}+x^5+1$	8 Bit Werte
CRC-32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$	Internet

Es ist offensichtlich, dass mit dem CRC-32 Standard die größte Sicherheit erreicht werden kann aber auch der höchste Aufwand verbunden ist.

Die Software-basierte Berechnung der Prüfsummen ist relativ aufwendig. Die Berechnung kann aber sehr effizient durch eine einfache Schieberegisterschaltung realisiert werden. In der Praxis wird diese Hardware-basierte Lösung fast immer eingesetzt.

18 Massenspeicher

18.1 Einführung

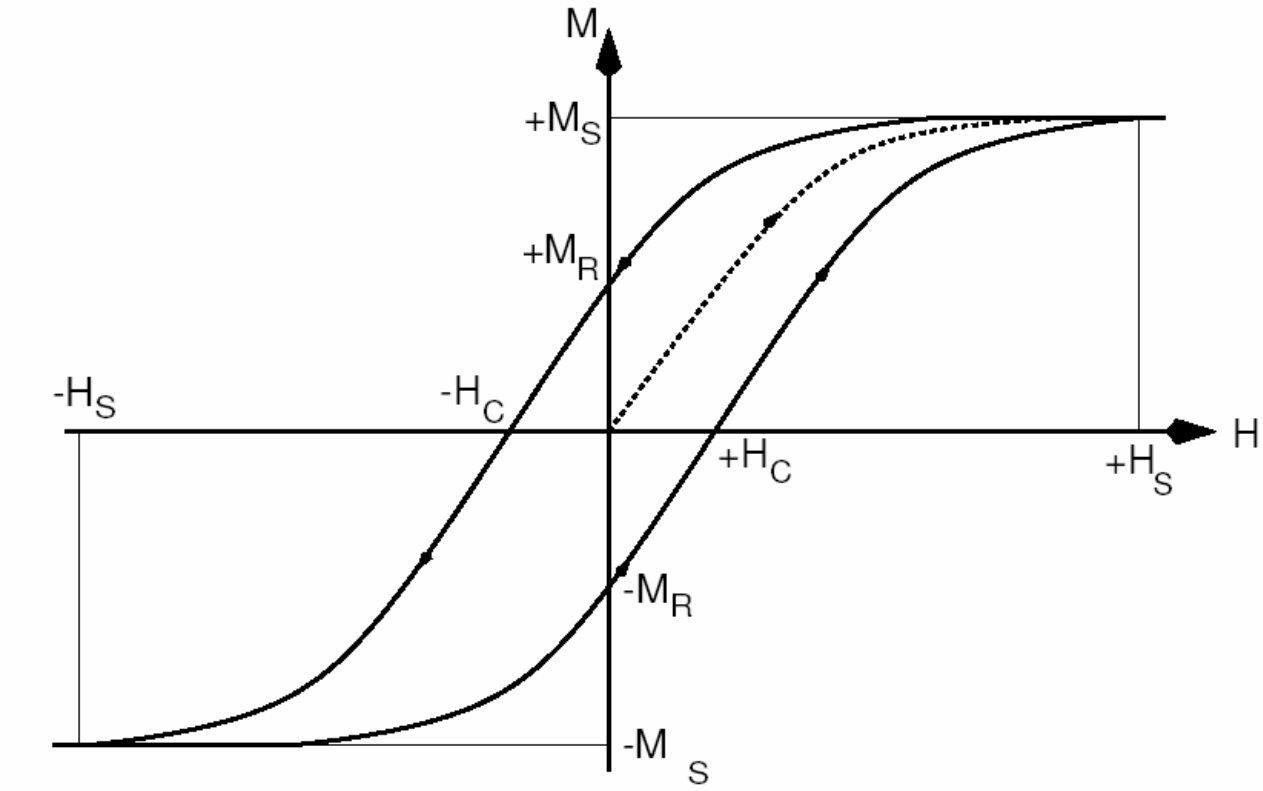
Für kleine Computer-Anwendungen reicht ein Prozessor mit Hauptspeicher (ROM und RAM) sicherlich aus. Möchte man jedoch große Datenmengen verarbeiten und permanent speichern, so müssen geeignete periphere (sekundäre) Massenspeicher bereitstehen.
Veraltet sind Lochstreifen und Lochkarten. Nur noch für sehr große Datenmengen und für die Archivierung und Datensicherung von Bedeutung sind Magnetbänder. In Spezialanwendungen werden z.T. relativ junge Technologien eingesetzt (z.B. Magnetblasenspeicher). Zunehmend wichtig werden optische Speichermedien (vorwiegend Platten).
Von großer Bedeutung sind Platten- und Diskettenlaufwerke. Diese magnetischen Massenspeicher sind in Rechnern aller Größenordnungen, vom Home- bis zum Supercomputer vertreten.
Die technologische Seite dieser Massenspeicher soll im folgenden näher behandelt werden.

18.2 Prinzip der magnetischen Aufzeichnung

18.2.1 Prinzip

Die bekannten Phänomene des Magnetismus sind der Diamagnetismus, der Paramagnetismus und der Ferromagnetismus. Auf dem Ferromagnetismus beruhen die aktuellen Verfahren zur magnetischen Aufzeichnung von Daten.
Diamagnetismus und Paramagnetismus sind nur mit äußerem magnetischen Feld zu beobachten. Beim Diamagnetismus, der wesentlich schwächer ist als der Paramagnetismus, wird das Magnetfeld durch das Einbringen des Materials abgeschwächt. Alle Substanzen sind diamagnetisch. Die Abschwächung ist materialspezifisch und recht klein. Paramagnetische Substanzen haben in ihrer Hülle mindestens ein ungepaartes Elektron, welches durch ein äußeres magnetische Feld ausgerichtet wird und so zu einer Verstärkung des Magnetfeldes führt.
Ferrormagnetismus hingegen kann auch ohne äußeres magnetisches Feld beobachtet werden. Bei ferromagnetischen Substanzen, wobei Eisen die vermutlich bekannteste ist, sind innerhalb kleiner Bereiche alle Atome vollständig einheitlich ausgerichtet. Diese Bereiche werden als

Domänen oder Weiß'sche Bezirke bezeichnet. Ferromagnetische Substanzen sind trotzdem nach außen normalerweise nicht magnetisch, da die Ausrichtung der Gebiete statistisch verteilt ist und daher das Material auf der makroskopischen Ebene unmagnetisiert wirkt. Wenn ein ferromagnetisches Material einem äußeren Magnetfeld ausgesetzt wird, richten sich die Gebiete nach dem äußeren Magnetfeld aus und das Material ist auch auf der makroskopischen Ebene magnetisch. Der Effekt bleibt auch nach der Abschaltung des äußeren Magnetfeldes erhalten.
Das Prinzip der magnetischen Datenspeicherung beruht auf dem Ferromagnetismus. Eine binäre Codierung erfolgt durch die Ausrichtung der Weiß'schen Bezirke in einer definierten Umgebung. Dies bedeutet, dass die maximale Auflösung durch die Größe der Weiß'schen Bezirke begrenzt ist. Um eine zuverlässige Speicherung der Daten erreichen zu können, darf die Ausrichtung nicht durch zufällige Einflüsse aufgehoben werden.



Hystereseschleife eine Ferromagnetikum

In der Abbildung ist die Hytereseschleife für ein Ferromagnetikum dargestellt. In der Darstellung ist die makroskopische Magnetisierung mit M und das äußere Magnetfeld mit H bezeichnet. Wie besprochen ist das Material im Ursprungszustand unmagnetisch, d.h. eine Magnetisierung M ist nicht messbar. Wird nun ein äußeres Magnetfeld H angelegt, so wird mit ansteigender Feldstärke eine Sättigung der Magnetisierung erreicht (M_s). Wird das äußere Feld anschließend abgeschaltet, bleibt eine Magnetisierung M_R erhalten, die als Remanenz bezeichnet wird. Das Material wurde also magnetisiert. Um die Magnetisierung aufzuheben muss gemäss Darstellung ein entgegen gesetztes Magnetfeld mit der Stärke $-H_C$, das als Koerzivität bezeichnet wird, angelegt werden. Wird das äußere Magnetfeld nun verstärkt wird eine entgegen gesetzte Magnetisierung erreicht, die ebenfalls eine Sättigung ($-M_S$) hat. Wir das äußere Magnetfeld nun abgeschaltet bleibt eine Magnetisierung $-M_R$ erhalten. Im magnetisierten Zustand hängt die Richtung der Magnetisierung daher von der Vergangenheit ab, d.h. welches äußere Feld zuletzt angelegt wurde. Dieses Phänomen wird als Hysterese bezeichnet und kann als ein Gedächtnis des Materials interpretiert werden. Exakt dieses Phänomen wird bei der magnetischen Speicherung ausgenutzt. Materialien mit hoher

Koerzivität und hoher Remanenz werden als magnetisch hart bezeichnet. Für die Herstellung von magnetischen Speichermedien mit hoher Speicherkapazität werden magnetisch harte Substanzen benötigt. Dies bedeutet, dass auch die Feldstärken zur Magnetisierung der Gebiete entsprechend hoch sein müssen. Magnetisch harte Substanzen können durch spezielle Legierungen hergestellt werden.

Eine weitere Eigenschaft ferromagnetischer Substanzen ist die Existenz der sogenannten Curie-Temperatur TC. Bei dieser Temperatur verschwinden die ferromagnetischen Eigenschaften des Materials schlagartig. Dies bedeutet insbesondere eine vollständige Aufhebung der Magnetisierung des Materials.

Zusammenfassend basiert das Prinzip der magnetischen Speicherung auf den ferromagnetischen Eigenschaften des Trägermaterials. Für die Speicherung der Daten werden die magnetischen Speichermedien in Gebiete eingeteilt, die jeweils ein Bit speichern. Mit der Ausrichtung des Gebietes wird die Wertigkeit des zugeordneten Bits codiert.

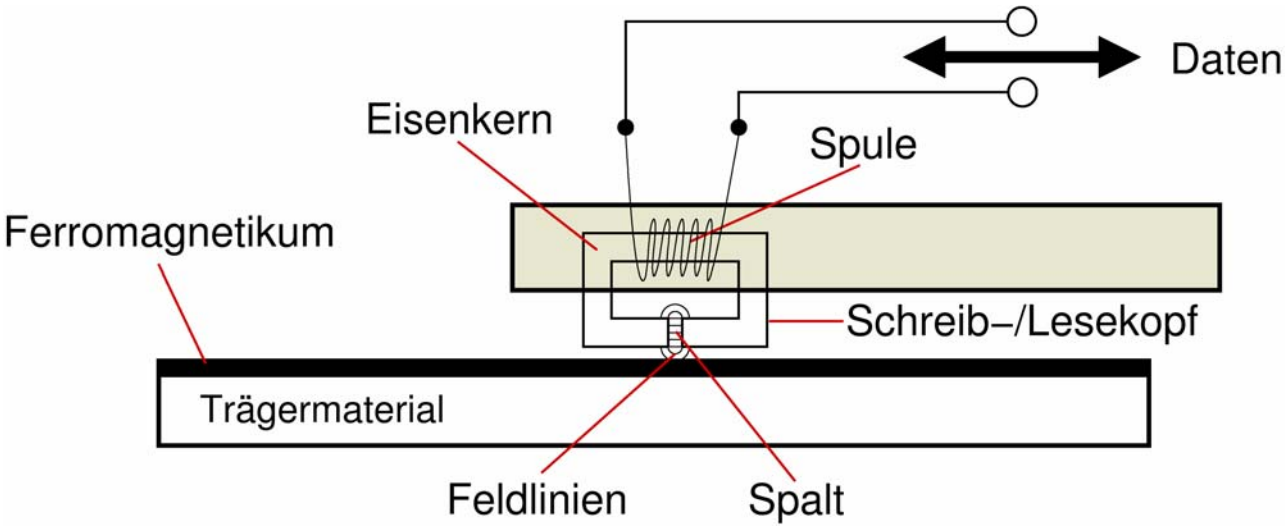
Schreiben und Lesen von Daten
Bei konstantem Stromfluss durch eine Spule wird ein Magnetfeld erzeugt. Mit Hilfe einer solchen Spule, die auch als Elektromagnet bezeichnet wird, können die Gebiete eines Speichermediums ausgerichtet werden. Auf diese Weise können Daten geschrieben werden. Für das Auslesen der Daten kommt ein physikalischer Effekt zum Einsatz, der als Induktion bezeichnet wird. Ein sich veränderndes Magnetfeld induziert eine elektrische Spannung. Wenn eine Spule über einem Medium bewegt wird, das unterschiedlich magnetisiert ist, wird daher in der Spule eine Spannung induziert. Durch eine Auswertung des Spannungsverlaufs können somit die geschriebenen Daten ausgelesen werden.

18.2.2 Speichermedien, Magnetbänder

Magnetbänder wie beispielsweise Tonbänder oder Streamer-Kassetten. Derartige Bänder werden für Aufgaben der Datensicherung auch heute noch verwendet. Die Justierung erfolgt eindimensional, indem das Band am Schreib-/Lesekopf vorbei geführt wird und dieser sich nicht bewegt. Im schlimmsten Fall muss das Band für einen Zugriff einmal umgespult werden. Diese Medien haben daher den Nachteil einer langen Zugriffszeit, da der Zugriff auf ein Bit bzw. auf eine Folge von Bits durch eine entsprechende Justierung des Bandes erfolgt, die konstruktionsbedingt aufwendig ist, da die Ansteuerung nur sequentiell erfolgen kann.

18.2.3 Speichermedien, Magnetplatten

Mit drehenden magnetischen Platten die beispielsweise zur Realisierung von Disketten oder Festplatten zum Einsatz kommen kann ein sehr schnelle Zugriff auf Daten realisiert werden. Dies hat den Grund, dass die Justierung zweidimensional erfolgt. Zum einen eine führt die Platte ständig eine Drehbewegung aus und zum anderen kann der Schreib-/Lesekopf mit Hilfe eines Zugriffarmes radial positioniert werden. Im schlimmsten Fall wird hier für den Zugriff eine volle Drehung des Mediums und eine vollständige lineare Bewegung des Schreib-/Lesekopfes benötigt. Beides kann in der Regel in einem Bruchteil einer Sekunde ausgeführt werden. Der Aufbau dieser Medien soll im folgenden etwas ausführlicher diskutiert werden.



Prinzipieller Aufbau einer Magnetplatte mit Schreib-/Lesekopf

In obiger Abbildung ist das Prinzip des Speicherns und Lesens von Daten auf einer Magnetplatte dargestellt. Der Schreib-/Lesekopf besteht aus einem kleinen Elektromagneten mit Eisenkern, der an einem Zugriffsarm befestigt ist. Der Eisenkern hat auf der Seite des Speichermediums einen Spalt aus dem die magnetischen Feldlinien beim Schreiben austreten. Mit den austretenden Feldlinien können Bereiche auf dem Speichermedium magnetisiert werden. Je kleiner der Spalt ist desto kleiner sind die Radien der austretenden Feldlinien. Der Radius der wirksamen Feldlinien nimmt auch mit dem Abstand zwischen Spalt und Speichermedium ab. Das Ziel ist daher den Spalt möglichst klein zu realisieren und den Kopf möglichst dicht am Speichermedium zu positionieren damit möglichst kleine Bereiche zur Speicherung eines Bit adressiert werden können. In Diskettenlaufwerken schleift der Kopf aus diesem Grund auf der Oberfläche der Diskette. Dies hat natürlich den Nachteil eines erhöhten Abnutzung der Oberfläche. Daher kommt dieses Prinzip bei Festplatten, die ständig in Bewegung sind, nicht zur Anwendung.

Das Trägermaterial ist beispielsweise bei Disketten mit einer Eisen- oder einer Kobaltverbindung beschichtet. Beim Schreiben und Lesen wird der radiale Ort des Zugriffs durch eine exakte Positionierung des Zugriffsarms mit Hilfe eines Stellmotors erreicht. Zusätzlich muss der Zeitpunkt des Zugriffs exakt auf die Drehposition des Mediums abgestimmt werden damit ein bestimmtes Gebiet adressiert werden kann.

Die Medien werden über Spuren organisiert, die unterschiedlichen Radien haben. Zwischen den Spuren liegen Bereiche die nicht magnetisiert sind. Bei Systemen mit doppelseitiger Beschichtung oder mehreren Platten (bspw. Festplatten) hat jede Seite der Platten einen eigenen Schreib-/Lesekopf. Die Spuren, die geometrisch auf den verschiedenen Platten gleich positioniert sind, werden typischerweise zusammengefasst und als Zylinder bezeichnet. Bei neueren Festplatten können mehrere Schreib-/Leseköpfe gleichzeitig aktiv sein. Die Drehposition des Mediums wird über Sektoren gekennzeichnet. Ein Sektor hat üblicherweise 512 Bytes. Um eine Überprüfung der Positionierung zu ermöglichen, wird vor jedem Sektor die Adresse des Sektors geschrieben. Da diese Information Speicherplatz benötigt, haben unformatierte Medien eine höhere Kapazität als formatierte. Die Speicherung von Adressdaten für Spuren, Zylinder und Sektoren wird als Low Level Formatierung bezeichnet.

Beispiel: Festplatten
Für aktuelle Festplatten wird meist Aluminium als Trägermaterial verwendet, da Platten mit diesem Material ihre Nenndrehzahl schneller erreichen können als Platten, die aus

beispielsweise Eisen gefertigt werden. Das Trägermaterial wird typischerweise mit Kobalt oder Eisenkeramiken als Ferromagnetikum beschichtet. Bei Kobalt, das heute meist verwendet wird, beträgt die Schichtdicke 0,1-0,2 Mikrometer. Die Beschichtung erfolgt durch Galvanisierung (Ionen werden aufgebracht und anschließend wird eine Spannung angelegt) oder Sputtern (Ionenbeschuss).
Für aktuelle Festplatten werden häufig Köpfe aus magnetoresistiven Materialien (MR-Köpfe) eingesetzt, die extrem leicht und nur wenige Mikrometer groß sind. MR-Köpfe werden mikromechanisch im Dünnfilm-Verfahren hergestellt. Der Abstand zu den Platten wird über Luftpolster erzeugt. Die Höhe des Luftpolsters hängt von der Form des Kopfes und vom Luftdruck ab. Auf der Zugspitze fliegt der MR-Kopf daher dichter an den Platten als im Flachland. Aus diesem Grund werden in den Datenblättern von Festplatten auch die Höhenlagen für den Einsatz spezifiziert.
Zur Positionierung der Schreib-/Leseköpfe wird bei Diskettenlaufwerken ein Schrittmotor verwendet. Dies bedeutet, dass die tatsächlich eingenommene Position für einen Schreib-/Lesevorgang nicht rückgekoppelt wird und daher auch nicht automatisiert korrigiert werden kann. Schrittmotoren müssen aber in regelmäßigen Abständen kallibriert werden und für thermische Veränderungen muss eine Kompensation berücksichtigt werden. Für moderne Festplatten werden hingegen Linearmotoren, die über einen Regelkreis gesteuert werden verwendet. Dies bedeutet, dass der Kopf für jeden Zugriff über eine Rückkopplung exakt positioniert wird. In einigen Modellen wird eine separate Plattenseite mit Servospuren als Referenz verwendet. Mit Hilfe der Servospuren wird die Information für die aktuelle Position geliefert.

18.3 Der Plattenstapel

Der Plattenstapel besteht aus magnetisierbaren Scheiben (Disks), auf denen die Daten gespeichert werden. Die Anzahl der Scheiben liegt zwischen 1 (Disk) und etwa 20.
Die Scheiben selbst können flexibel (Disk) oder massiv sein (beschichtete Alu-Platte).
Der Plattenstapel kann fest installiert (fixed Disk) oder austauschbar (Disk, Wechselplatte, removable disk) sein.
Übliche Größen für Disketten sind 3 ½“ und 5 ¼“ (weniger gebräuchlich 3“ und 8“), übliche Größen für Platten sind 5 ¼“, 8“ und 14“ (3 ½“).
Auf dem Plattenstapel ist eine Index-Kennung angebracht (z.B. ein Loch, das mit einer Lichtschranke abgetastet wird). Diese wird benötigt, um einen definierten Anfangspunkt auf dem Kreisumfang zu haben.

18.3.1 Hardwareaspekte

18.3.1.1 Der Antriebsmotor

Der Antriebsmotor treibt den Plattenstapel an. Die Rotationsgeschwindigkeiten liegen Zwischen 300 U/Min für Diskettenlaufwerke und etwa 5.000 U/Min (schnelle Platten-Laufwerke). Oft ist eine (Kurzschluß-)Bremse eingebaut, die den Plattenstapel beim Abschalten des Laufwerkes möglichst schnell zum Stillstand bringt.

18.3.1.2 Die Magnetköpfe

Die Platten werden von den Magnetköpfen gelesen bzw. beschrieben. Die Köpfe bewegen sich in einem geringen Abstand (bis herab zu 0,3 Mikrometer) über der Platte. Je kleiner der Abstand ist, desto höher kann die Informationsdichte auf der Platte sein.
Dieser Abstand kann durch unterschiedliche Verfahren erreicht werden. Bei Disketten schleifen die Köpfe auf dem flexiblen Medium, bei Festplatten baut sich ein Luftkissen zwischen den schnell rotierenden Platten und den Köpfen auf. Die Köpfe sind in diesem Fall als Flugkörper ausgebildet.

18.3.1.3 Der Kopfträger und Steppermotor

Die Köpfe sind auf einem Träger gelagert. Dieser kann von dem Steppermotor radial bewegt werden. Der Motor ist meist ein Voice-Coil-Antrieb (Tauchspule in einem Magnetfeld), bei älteren oder langsamen Laufwerken auch ein Schrittmotor (Spindel- oder Stahlbandantrieb).
Ähnlich der Index-Kennung dient ein Spur 0 Indikator dazu, die Kopfposition zu erkennen. Nach dem Einschalten werden die Köpfe so lange nach außen bewegt, bis der Indikator anspricht. Damit ist eine definierte Kopfposition festgelegt.
Diskettenlaufwerke haben einen Kopflademagneten. Dieser hat die Aufgabe, die Köpfe beim Schreiben und Lesen gegen die Diskette zu drücken.
Ein ähnlicher Mechanismus ist auch bei hochwertigen Plattenlaufwerken vorhanden. Hier werden die Köpfte automatisch entladen, wenn die Umdrehungsgeschwindigkeit des Plattenstapels absinkt. Durch diese Maßnahme wird der berüchtigte „Head-Crash“ vermieden, ein ungewolltes gewaltsames Aufsetzen des Kopfes auf der Platte. Durch das Aufsetzen werden Kopf und Platte zerstört (spanabhebende Datenverarbeitung).
Bei Plattenlaufwerken, die diese Fähigkeit nicht besitzen, ist oft eine spezielle Parkposition vorgesehen, auf der die Köpfe beim Abschalten des Laufwerks landen können.

18.3.1.4 Plattenorganisation

Ein Plattenstapel wird wie folgt unterteilt:

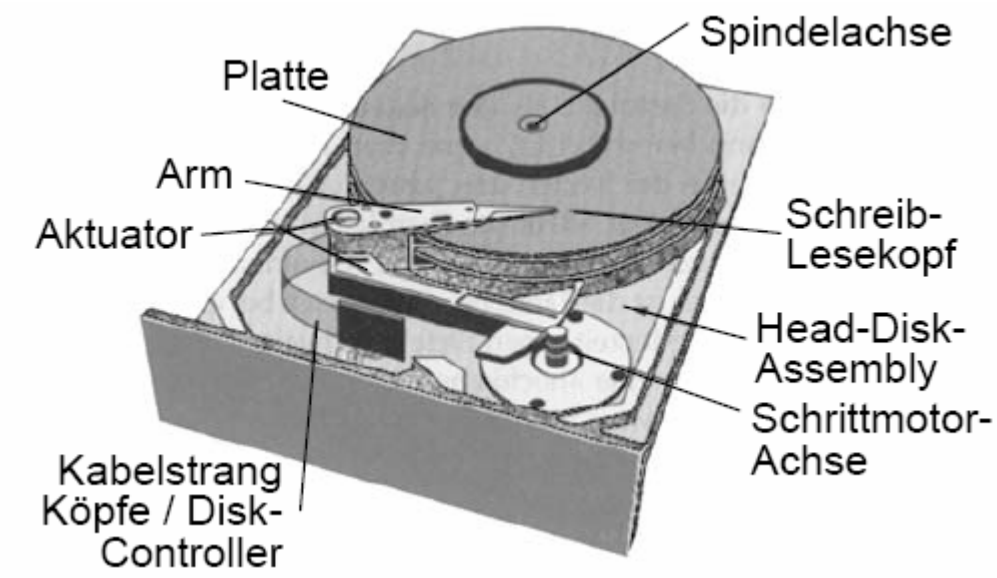
Jeder Kopf arbeitet auf einer Oberfläche (surface). Jede Oberfläche ist in konzentrische Kreise unterteilt, die Spuren (tracks) genannt werden.
Die äquivalenten Spuren auf allen Oberflächen heißen Zylindern (nach dem mathematischen Modell).

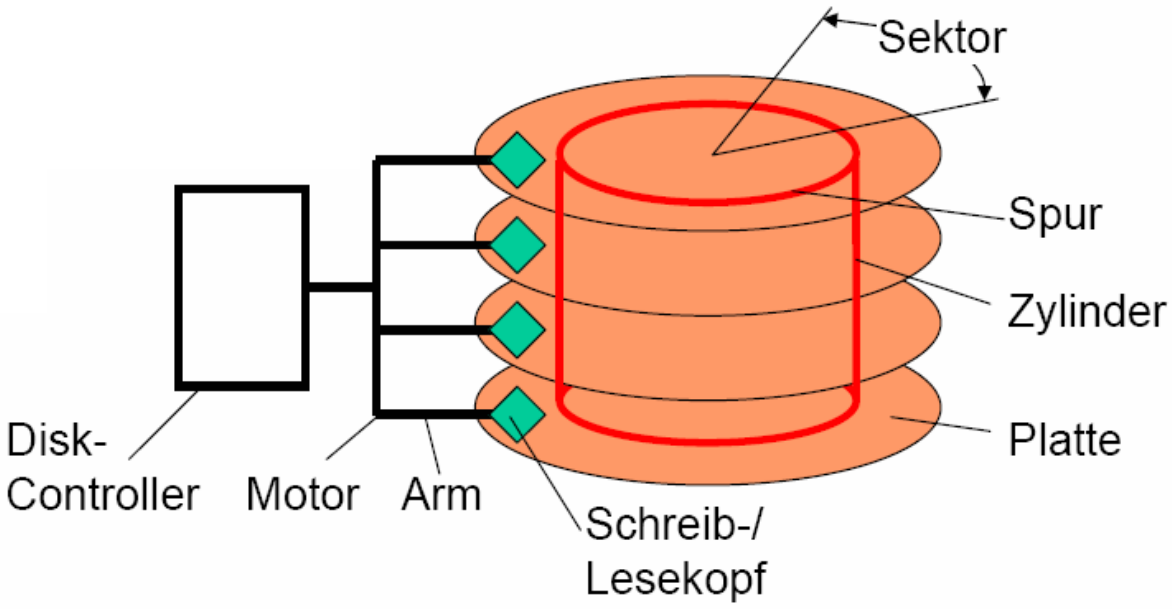
18.3.1.5 Zylinder und Spur

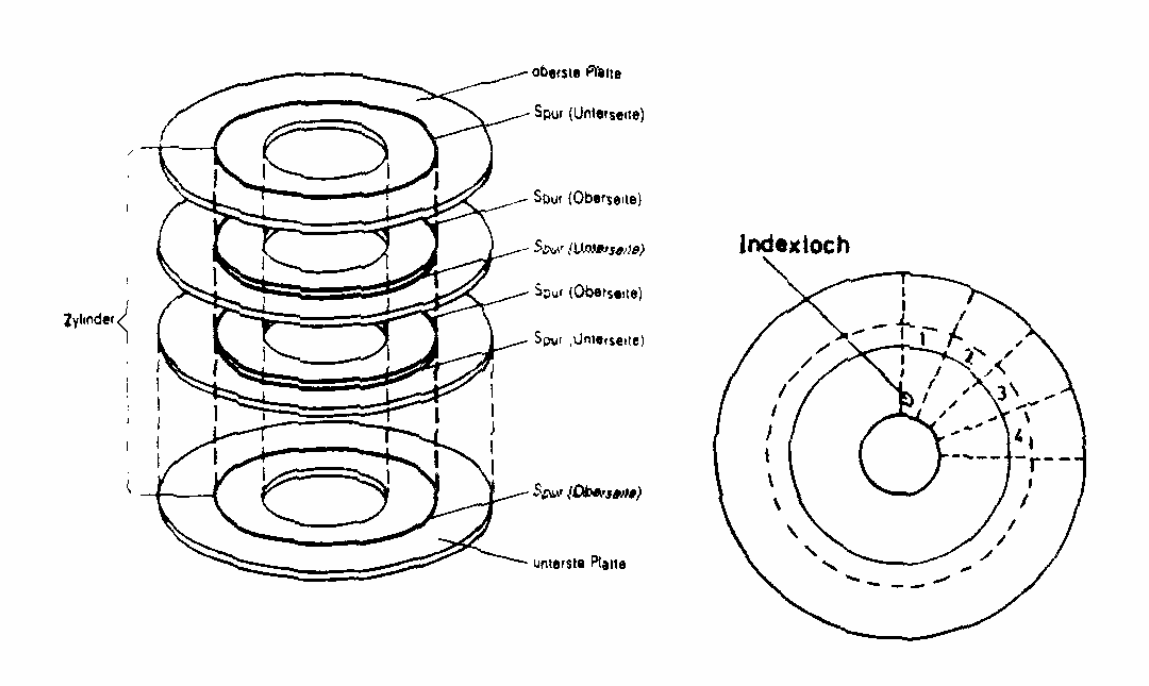
Die Tracks sind in Sektoren eingeteilt. Ein Sektor ist zugleich die kleinste Einheit, die vom Computer gelesen oder beschrieben werden kann. Ein Sektor besteht aus einer festen Anzahl von Bytes. Gebräuchliche Sektor-Größen sind Zweier-Potenzen ab 128 Bytes (256, 512, …). Für die Sektoren werden ebenfalls Indikatoren benötigt. Diese werden meist beim Formatieren magnetisch aufgeschrieben (soft-sektoriert).
Vereinzelt noch in Gebrauch sind hard-sektorierte Platten. Hier wird die Einteilung in Sektoren ähnlich wie bei der Indexkennung vorgenommen.

18.3.1.6 Physikalischer Aufbau von Festplatten

Beispiel: Seagate Cheetah 36
• 3,5 Inch Disk
• 36,4 GByte Kapazität
• 10.000 Umdrehungen/Minute
• 18,3 bis 28 MByte/s interne Datentransferrate
• 9.772 Zylinder (Spuren)
• 71.132.960 Sektoren insgesamt
• Mittlere Zugriffszeiten:
Lesen 5.2 ms,
Schreiben 6,0 ms







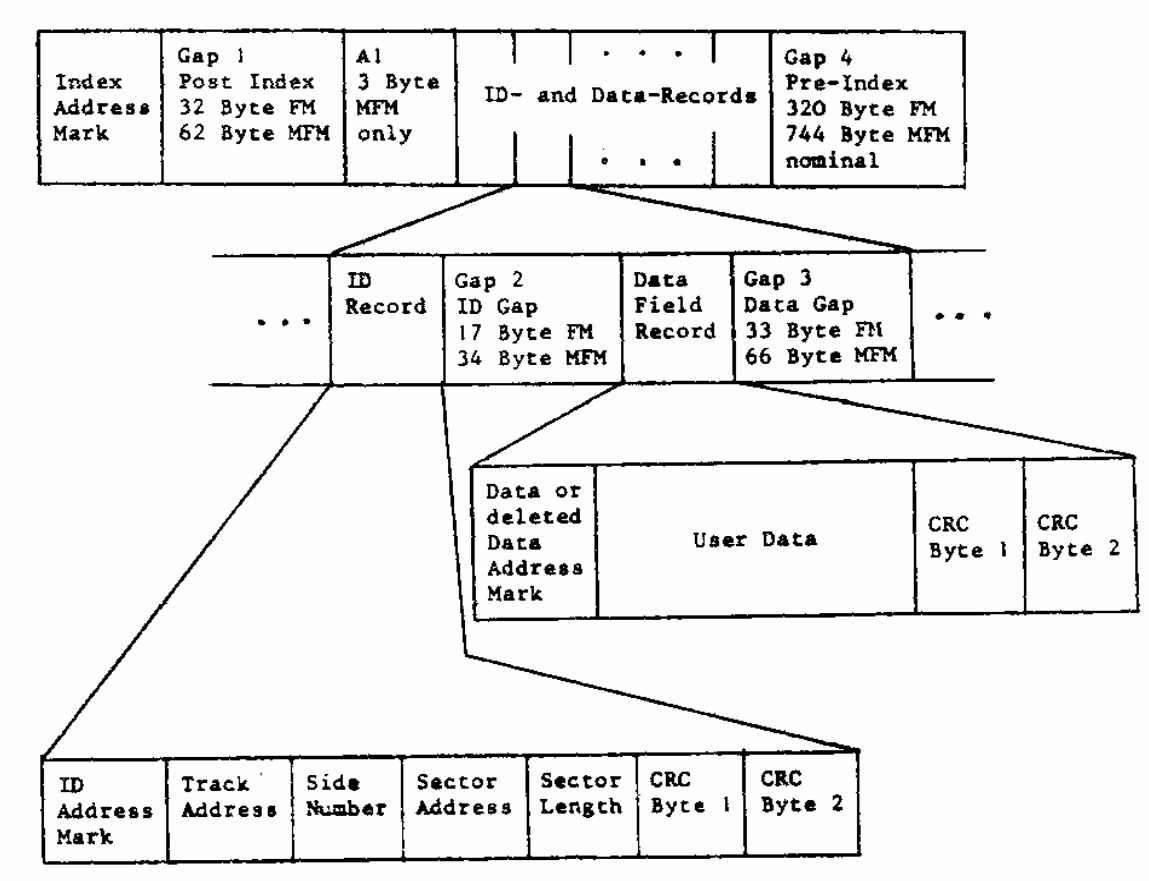
18.3.1.7 Formatierung

Unter Formatierung versteht man die Aufteilung der Platte in Nutz- und Kontroll-Informationen sowie Informationslücken. Dies geschieht durch einen besonderen Schreibvorgang (meist nur einmal im Leben einer Platte).

Ein typischer Sektor ist wie folgt aufgebaut:

- Sektor-Identifikation
    - Kopf-Nummer (Oberfläche)
    - Zylinder-Nummer
    - Sektor-Nummer

  Check ((z.B. Cyclic-Redundancy-Check CRC), wird später besprochen)
- Sektor-Daten
    - Nutzdaten
    - Check



Zwischen Identifikationsfeld und Datenfeld ist eine Lücke vorhanden, ebenso nach dem Datenfeld. Die Lücken sind beim Schreiben eines Sektors nötig. Beim Umschalten zwischen Lesen (des ID-Feldes) und Schreiben (des Datenfeldes) benötigt das System einige Zeit, bis ein stabiles Schreiben möglich ist (Einschwingvorgänge). Eine genaue Synchronisation mit den bereits vorhandenen Daten ist somit nicht möglich. Außerdem werden vor dem ID- bzw. Datenfeld noch einige Bytes geschrieben, damit die für das Lesen erforderliche Synchronisation (z.B. durch eine PLL) erreicht wird (Synchronisationsbytes). Hierin enthalten sind spezielle Kennungen, die angeben, ob es sich hierbei um ein ID-Feld bzw. und ein Datenfeld handelt.
Oft wird bei der Formatierung noch eine Indexkennung auf jede Spur geschrieben. Diese hat die gleiche Funktion wie das Index-Loch. Vor und nach der Indexkennung sind ebenfall Lücken vorhanden, ebenso am Ende einer Spur.
Bei größeren Platten wird oft eine sogenannte „Servo“-Oberfläche ausschließlich mit Kontroll-Informationen beschrieben. Neben den Aufgaben der Formatierung können hierdurch weitere Probleme gemeistert werden. Z.B. dehnt sich die Platte mit steigender Temperatur aus. Mit Hilfe der Servo-Informationen als Referenz kann dieser Effekt durch genaues Positionieren der Köpfe kompensiert werden.

18.3.1.7.1 Low Level Formatierung (Hersteller)

• Platz auf der Festplatte wird in Spuren (Tracks) und Sektoren mit Interleaving aufgeteilt.
• Bei aktuellen Festplatten erfolgt die Formatierung nach dem Zone-Bit Recording – Verfahren.
• Beschädigte Bereiche der Platten werden ausgespart.
• Die sogenannte Geometrie (vom Hersteller erzeugte Struktur) der Festplatten wird nach außen, d.h. vor dem Anwender, verborgen.

18.3.1.7.2 High Level Formatierung (Anwender)

• Jeder Zylinder hat aus der Sicht des Anwenders gleich viele Sektoren.
• Bei PCs ist jeder Sektor 512 Byte groß.
• Dateisysteme fassen die Sektoren zu Clustern zusammen, deren Größe vom Dateisystem und der Partitionsgröße abhängen. Beispiel:
FAT16 (FAT = File Allocation Table) DOS nutzt einen 16-Bit – Adressraum
-> 65.536 Einheiten können verwaltet werden:
Einheit = Sektor: 512 KByte * 65.536 = 32 MByte
Einheit = Cluster zu 4, 8, 16, 32 oder 64 Sektoren:
z.B. 64 * 0,5 MByte * 65.536 = 2 Gbyte

18.3.1.8 Adressierung von Daten auf Festplatten

Zwei Verfahren:
CHS = Cylinders, Heads, Sektors, (vergl.: Stadt, Straße, Hausnummer)
Beispiel:
Durch das BIOS ist die Festplattengröße auf 8 GByte beschränkt.
Das Bios verwendet 24 Bit zur Adressierung:
10 Bit für Zylinderzahl (max. 1024, „1024 Zylinder Problem“)
8 Bit für Heads (max. 256)
6 Bit für Sektoren (max. 64)

LBA = Logical Block Address, (vergl.: Alle Häuser werden einfach durchnummeriert)
-> Wird bei aktuellen Festplatten verwendet.
ATA-Spezifikationen: ATA 100: ≤ 137 GByte
ATA 133: 48 Bit Adressierung ≥ 137 GByte
SCSI: Controller verwenden eigenes BIOS, haben aber ähnliche Probleme

18.3.1.9 Master Boot Record und Partitionen

Master Boot Record (MBR)
• Boot-Vorgang: BIOS sucht nach dem MBR, lädt das darin enthaltene Programm und bringt es zur Ausführung.
Programm-Größe: 446 Byte
Programm: Betriebssystem-Kern oder Boot-Loader/Manager (z.B. bei WinXP stehen in Datei C:\boot.ini die Anweisungen für den Boot-Loader)
• Der MBR hat einen festen Platz auf der Festplatte:
Zylinder 0, Head 0, Sektor 1
• Das Ende des MBR wird mit 55AAH bezeichnet. Der MBR enthält den Boot-Loader und vier Partitionstabellen.
Partitionen:
• Partitionen bilden „Virtuelle Festplatten“.
• Die erste Partition beginnt bei: Zylinder 0, Head 1, Sektor 1
(etwas Platz bleibt zwischen MBR und erster Partition frei. Disk-Manager nutzen oft den freien Platz für ihre Zwecke.)
• Jede Partition hat eine eigene Partitionstabelle. Sie ist 64 Byte lang und besteht aus 4 * 16 Byte – Einträgen (z.B.: Partition bootfähig, Größe usw.).

18.3.1.10 Plattenkapazität

Die Kapazität eines Laufwerks kann formatiert oder unformatiert angegeben werden. Die formatierte Kapazität gibt nur die Menge der Nutz-Informationen an. Die unformatierte enthält noch die Kontroll-Informationen und Lücken. Die Kapazität kann für einen Sektor, eine Spur, einen Zylinder, eine Oberfläche oder die gesamte Platte angegeben werden.

18.3.1.11 Zugriffszeit

Für den Zugriff ein einen beliebigen Sektor sind mehrere Aktionen notwendig.

Zuerst einmal muss das Laufwerk ausgewählt werden (Mikrosekundenbereich). Dieses muss evt. erst anlaufen, d.h. die Platte in Bewegung setzen (vorwiegend bei Diskettenlaufwerken). Die dazu benötigte Zeit liegt in der Größenordnung von einer Sekunde.
Als nächstes muss der richtige Zylinder angefahren werden. Dies ist Aufgabe des Stepper-Motors. Hier sind 3 Zeiten von Bedeutung:

- Spur zu Spur (benachbarte Spuren)
- Mittlere Positionierzeit (average)
- Maximale Positionierzeit (maximum) zwischen dem innersten und äußersten Zylinder

Nach Anfahren des Zylinders muss sich das Positioniersystem beruhigen (settling time).
Die Auswahl des richtigen Kopfes geschieht im Mikrosekundenbereich und kann gegenüber den anderen Zeiten vernachlässigt werden. Bei Diskettenlaufwerken muss der Kopf noch geladen werden (load time) bzw. vor dem Positionieren entladen werden (unload time).
Schließlich muß der richtige Sektor gelesen werden. Im günstigsten Fall geschieht dies sofort, im ungünstigsten Fall muss eine Umdrehung abgewartet werden. Im Mittel ergibt sich eine Wartezeit (latency) von einer halben Umdrehung.
Nachdem der richtige Sektor gefunden wurde, kann der Sektor übertragen werden. Hierzu wird ebenfalls Zeit benötigt; diese Zeit hängt von der Sektorgröße, der Aufzeichnungsdichte und der Umdrehungsgeschwindigkeit ab.

18.3.1.12 Weitere technische Daten

Weitere Merkmale von Plattenlaufwerken werden durch folgende technische Daten charakterisiert:
Die Aufzeichnungsdichte – gemessen in bits per inch (bpi) – gibt an, wie dicht die Informationen auf einer Spur stehen. Sie errechnet sich aus dem Quotienten von unformatierter Spurkapazität und Spurumfang. Erhalten alle Spuren einer Platte dieselbe Sektoranzahl (und sind die Sektoren gleich groß), so steigt die Aufzeichnungsdichte von den äußeren Spuren zu den inneren Spuren hin an (die Spurkapazität bleibt konstant, der Spurumfang wird kleiner).

Die Spurdichte – gemessen in tracks per inch (tpi) – gibt die radiale Dichte der Informationen an.

Die Umdrehungsgeschwindigkeit (U/Min) ist selbsterklärend.

Die Übertragungsrate gibt an, mit welcher Geschwindigkeit die Daten gelesen bzw. geschrieben werden.

Das Interface Laufwerk – Computer

18.3.1.13 Die Elektronik

Dieser Teil enthält die notwendigen Schaltungen zum Betreiben des Laufwerks.
Die Kontrolle des Computers über die Plattenlaufwerke erfolgt über Plattencontroller.
Die Schnittstellen zwischen Computer und Controller können verschieden ausgelegt sein. So kann z.B. die CPU ein Diskettenlaufwerk direkt ansteuern (minimaler Hardware-Aufwand), oder der Controller kann selbst ein intelligentes System sein, welches eigenständig die Plattendaten puffert. Mit dieser Methode werden Echtzeitanforderungen beim Zentralcomputer vermieden.
Die Schnittstelle des Controllers zum Laufwerk wird durch den physikalischen Aufbau der Platte bestimmt und ist hierdurch im wesentlichen festgelegt. Die Bilder zeigen typische Interfaces zwischen Laufwerk und Controller.

Im einzelnen kann man folgende Signalgruppen unterscheiden.

18.3.1.13.1 Stepper-Kontrolle

Zur Kopfpositionierung werden 2 Signale verwendet. Direction gibt die Richtung an (nach innen oder außen); Step liefert Impulse zur relativen Positionierung.
Bei Plattenlaufwerken können diese Impulse oft direkt hintereinander folgen (Abstand Etwa 1 Mikrosekunde, Burst-Mode genannt). Diese werden von einem Laufwerk-internen Mikroprozessor mitgezählt. Der Mikroprozessor steuert dann den Steppermotor. Bei dieser Variante ist eine Rückmeldung an den Computer erforderlich (seek complete).

18.3.1.13.2 Schreib- und Leseleitungen

Hier findet die eigentliche Datenübertragung statt. Bei langsamen Laufwerken (Disketten) werden die Daten direkt übertragen (als TTL-Signale), bei schnellen Datenraten als Differenzsignale. Die Signale selbst sind keine binären Sequenzen mehr, sondern eine Folge von Pulsen. Diese Pulse werden dann auf der Platte als Flusswechsel abgespeichert. Hierzu erforderlich sind Kodierer und Dekodierer sowie Verstärker. Gebräuchliche Aufzeichnungs-Verfahren sind FM, MFM und RLL (2,7). Näheres hierzu finden Sie im Kapitel Kodierungs-verfahren.

18.3.1.13.3 Laufwerkauswahl

In der Regel werden an einem Platten-Controller mehrere Laufwerke betrieben, so dass eine Laufwerkauswahl (drive select) erfolgen muss. Diese kann durch eine Leitung pro Laufwerk oder auch binärkodiert erfolgen.

18.3.1.13.4 Kopfauswahl

Mit diesen Signalen wird binärkodiert der Kopf ausgewählt, der die Daten schreiben oder lesen soll.

18.3.1.13.5 Weitere Signale

- Spur 0 Indikator
- Index
- Write Protected
- Motor on (vorwiegend bei Diskettenlaufwerken)
- Write Gate (Schreibfreigabe
- Reduced write current (bei manchen Laufwerken muß für die inneren Spuren der Schreibstrom reduziert werden, da andernfalls das Medium übersteuert würde)

- Ready (Laufwerk bereit)

18.3.2 Kodierungsverfahren

Das Schreiben der Datenbits auf die Platte erfolgt nicht direkt, sondern über eine Kodierung. Der Grund hierfür wird später anhand der nachstehenden Kodierungen selbst erläutert.

18.3.2.1 NRZ und NRZI

Das einfachste Verfahren ist das direkte Aufschreiben auf das magnetische Medium. Eine logische 1 entspricht einer Magnetisierung in der einen Richtung, eine logische 0 einer Magnetisierung in der anderen. Unmagnetisierte Stellen sind nicht vorhanden; daher kommt auch der Name dieses Verfahrens (NRZ – Non Return to Zero).
Dieses Verfahren hat entscheidende Nachteile:
Beim Lesen detektiert der Kopf nämlich nicht den magnetischen Fluss direkt, sondern nur die Änderungen. Falls somit irgendwo eine Flussänderung fehlt, werden alle folgenden Bits invertiert gelesen.
Aus diesem Grund hat man ein modifiziertes Verfahren entwickelt (NRZI). Hier wird ein vorhandener bzw. fehlender Flusswechsel als logische 1 bzw. 0 kodiert.
Ein weiterer Nachteil macht sich bei langen Intervallen ohne Flusswechsel bemerkbar (viele logische Nullen): die fehlende Synchronisation. Durch Abweichungen von der Normgeschwindigkeit können Bits hinzugefügt oder gelöscht werden. Abhilfe schaffen hier Kodierungen, aus denen sich der Schreib-Takt ableiten lässt.

18.3.2.2 RLL-Kodierungen

Die Taktsynchronisation geschieht durch Hinzufügen von zusätzlichen Flusswechseln. Diese Kodierungen sind so ausgelegt, dass nach einer bestimmten Zeitspanne ohne Flusswechsel auf jeden Fall ein solcher erzwungen wird (RLL – Run Length Limited).

18.3.2.3 FM-Kodierung

Das einfachste RLL-Verfahren ergibt sich, wenn vor jedem Datenbit ein Flusswechsel generiert wird. Jedem Datenbit entsprechen also zwei physikalische Bits, ein Taktbit und das eigentliche Datenbit:

logisch	:	0	1
physikalisch	:	10	11

Einem gesetzten physikalischen Bit entspricht wieder ein Flusswechsel. Beim FM-Verfahren erzeugt also mindestens jedes zweite physikalische Bit einen Flusswechsel. Hierdurch kann sehr leicht synchronisiert werden. (Die von der Formatierung bekannten Adress- und Data-Marks werden übrigens durch fehlende Taktbits charakterisiert.)

18.3.2.4 MFM-Kodierung (Modified Frequency-Modulation)

Vergleicht man die FM-Kodierung mit der NRZ- oder NRZI-Kodierung, so fällt auf, dass relativ viele Flusswechsel pro Datenbit generiert werden. Der minimale Abstand bestimmt aber direkt die erreichbare Informationsdichte mit. Das magnetische Medium wird also gegenüber der NRZ-Kodierung nur zur Hälfte ausgenutzt.
Haben wir beim Übergang von der NRZI- zur FM-Kodierung gefordert, dass innerhalb einer Zeitspanne mindestens ein Flusswechsel auftritt, so legen wir jetzt zusätzlich eine minimale Zeitspanne zwischen zwei Flusswechseln fest. Kodiert man beispielsweise so, dass zwischen zwei physikalischen 1-Bits (Flusswechsel) mindestens 1 und maximal 3 physikalische 0-Bits liegen, ergibt sich die MFM-Kodierung:

logisch	:	10	1	0
physikalisch	:	0100	01	10

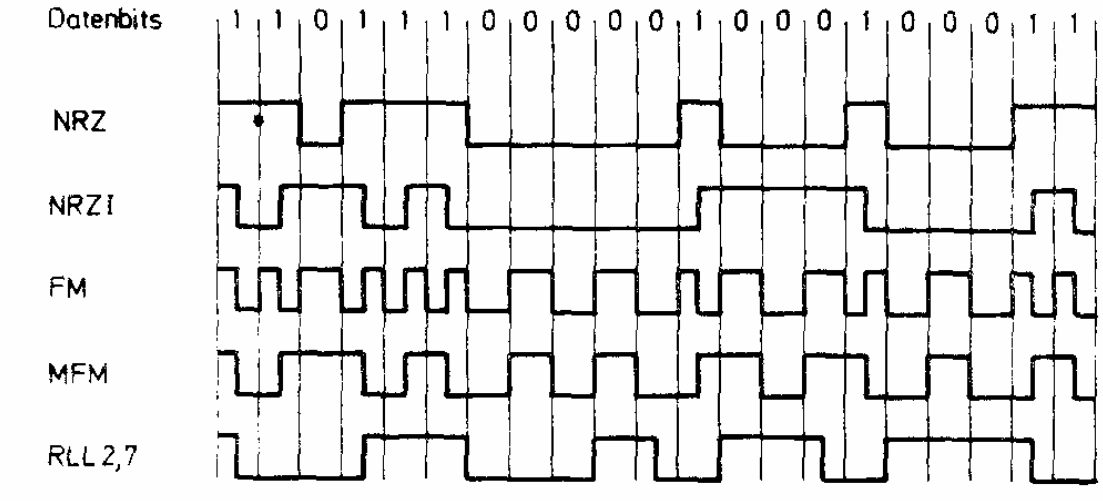
Die Datenbits sind hierbei mit der zuerst in der Tabelle auftretenden Kodierung zu übersetzen (die Folge 01011 wird also in 0 10 1 1 zerlegt und nicht in 0 1 0 1 1).
Wie man sieht, wird die Anzahl der Flusswechsel erheblich reduziert und ein minimaler Abstand garantiert. Gegenüber der FM-Kodierung ergibt sich wieder eine Verdoppelung der Informationsdichte (d.h. so gut wie die NRZ-Verfahren, aber ohne deren Nachteile).

18.3.2.5 Andere RLL-Verfahren

Natürlich lässt sich der Abstand zwischen zwei Flusswechsel auch anders feststellen als bei der MFM-Kodierung. Hier sind viele Varianten und Kodierungen möglich.
Weit verbreitet ist heute die RLL 2,7-Kodierung. Hier liegen mindestens 2 und maximal 7 physikalische 0-Bits zwischen zwei Flusswechsel:

logisch	:	11	10	000	010	011	0010	0011
physikalisch	:	1000	0100	000100	100100	001000	0010010	00001000

Man muss übrigens nicht immer einem Datenbit zwei physikalische Bits zuordnen. So wählte ein Hersteller ein Verhältnis von 4 Datenbits zu 6 physikalischen Bits in einer RLL 1,7-Kodierung. Entscheidend für die Informationsdichte ist jedoch nicht nur das Verhältnis zwischen Daten- und physikalischen Bits, sondern auch der Abstand zwischen zwei Flusswechsel. Mit der oben vorgestellten RLL 2,7-Kodierung ergibt sich z.B. gegenüber der MFM-Kodierung eine um 50 Prozent höhere Informationsdichte.



18.3.3 Logische Gliederung einer Platte

Bisher wurde nur die physikalische Aufteilung von Platten besprochen. Ein Sektor kann durch Angabe von der Sektornummer innerhalb einer Spur, Zylindernummer, Oberflächennummer und Plattennummer adressiert werden.
Für die Verwaltung von Daten ist jedoch eine Organisation auf höherer Ebene wünschenswert. In diesem Zusammenhang haben sich verschiedene Begriffe gebildet, die im folgenden näher erläutert werden sollen.

18.3.3.1 Dateien

Unter einer Datei (Dataset, File) versteht man – inhaltlich zusammenhängende – Daten, die gemeinsam organisiert werden. Eine Datei hat einen Namen, unter dem sie angesprochen wird.

Die Datei selbst kann unterschiedlich organisiert sein. Am einfachsten ist eine Folge von Sektoren. Hier ist noch ein Bezug zur physikalischen Platte vorhanden. Einfache Betriebssysteme (z.B. CP/M) unterstützen nur diese Art von Datei.

Eine unstrukturierte Datei ist als Byte-Folge aufzufassen. Dies ist aus Sicht des Programmierers die einfachste Zugriffsart.

Gebräuchlich ist die Aufteilung einer Datei in Sätze (Records). Die Zugriffe erfolgen dabei immer satzweise, d.h. das Betriebssystem (bzw. das Anwender-Programm) muß die physikalischen Sektoren in logische Sätze umstrukturieren. Die Sätze können entweder alle gleich lang sein (fixed record), oder unterschiedlich (variable record).

18.3.3.2 Directory

Ein Directory stellt ein Inhaltsverzeichnis für Dateien dar. Hier werden Kontrollinformationen über Dateien abgelegt, u.a. die Datei-Namen selbst. Directories können oft hierarchisch aufgebaut werden, d.h. ein Verzeichnis kann weitere Unter-Verzeichnisse haben (Sub-Directory). Alle Directories bilden somit einen Baum. Die Wurzel dieses Baumes wird Root-Directory (Haupt-Verzeichnis) genannt. Jede Platte hat genau ein solches Root-Directory.

18.3.3.3 Volume

Der Begriff Volume ist zweideutig. Man unterscheidet physikalisches und logisches Volume. Ein physikalisches Volume ist z.B. ein Plattenstapel, eine Diskette oder auch ein Band. Ein logisches Volume ist das, was der Computer als eine Einheit betrachtet.
Ein logisches Volume kann mehrere physikalische beinhalten (z.B. mehrere Bänder für die Datensicherung einer Platte). Umgekehrt kann ein physikalisches Volume auch mehrere logische enthalten (z.B. die Unterteilung eines Plattenlaufwerks in mehrere logische Einheiten; in diesem Fall spricht man auch von den Partitions einer Platte).
Notwendig für die Zuordnung vom physikalischen zum logischen Volume ist ein Verzeichnis hierzu (z.B. eine Partition-Table). Nur im einfachsten (aber häufigen) Fall einer direkten Zuordnung kann diese entfallen.

Auf einem logischen Volume sind neben dem Root-Directory noch weitere besondere Bereiche. So wird die Zuordnung der Sektoren zu den Dateien oft nicht in den Directories selbst abgelegt, sondern separat in einer File Allocation Table (FAT).
Beim Urladen eines Computers wird auf die Platte direkt zugegriffen, d.h. unter Umgehung der Directories und der FAT. Dieser Bereich wird Boot-Bereich (Boot-Sektoren, Boot-Spuren, …) genannt.

18.3.3.4 Katalog

Ein Katalog kann als Super-Directory bezeichnet werden, da ein Katalog Dateien verschiedener Platten (und Bänder) gemeinsam verwaltet. Mit Hilfe eines Katalogs braucht beim Zugriff auf eine Datei die Platte nicht mehr angegeben werden, da diese aus dem Katalog-Eintrag automatisch bestimmt wird.
Mit Hilfe eines Katalogs ist es auch möglich, eine Datei über mehrere Volumes zu verteilen. Dies ist bei sehr großen Dateien von Vorteil.

18.3.4 Zugriffsverfahren für Massenspeicher

Die Plattendaten können auf verschiedene Weisen angesprochen werden. Je nach Zweck und Organisation der Daten erweisen sich unterschiedliche Zugriffsarten als sinnvoll.

18.3.4.1 Programmed I/O (PIO)

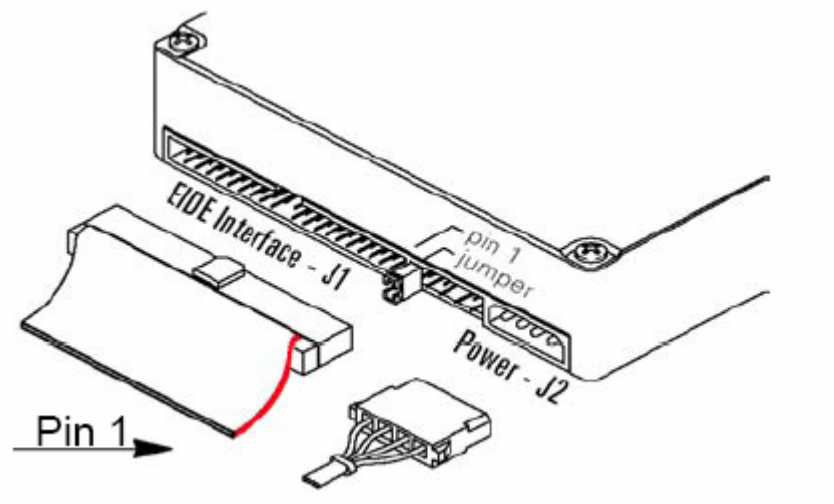
• CPU führt die Ein-/Ausgabe von oder zu einem Port aus.
• Einsatz bei virtueller Adressierung, da die Adressumsetzung von der MMU übernommen wird.

18.3.4.2 DMA-Transfer (Direct Memory Access)

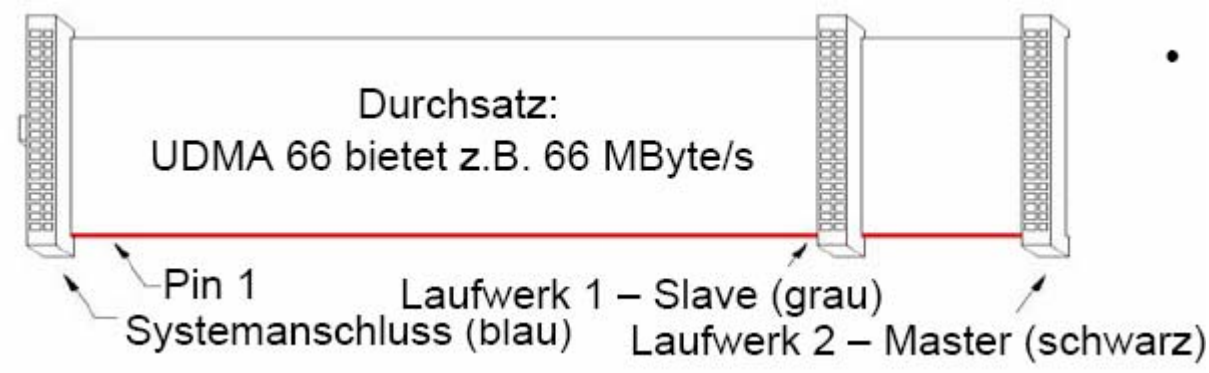
Ziel:
• Entlastung der CPU vom Datenaustausch zwischen Hauptspeicher und Massenspeicher.
Prinzip:
• Spezieller Ein-/Ausgabeprozessor übernimmt den Datenaustausch.
Ablauf:
• Prozessor übergibt die Startadresse der Daten, das Ziel der Übertragung und die Anzahl der zu übertragenden Daten an den DMABaustein.
• Der DMA-Baustein wickelt den weiteren Datenverkehr eigenständig ab.

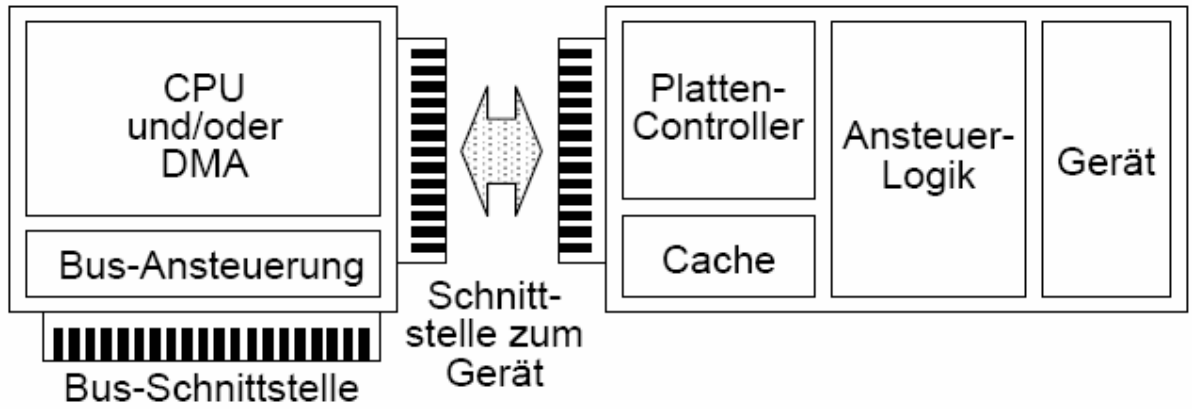
18.3.4.3 Anschlusstechnik für Festplatten und CD/DVD

EIDE – Anschluss von Festplatten oder CD/DVD



IDE bzw. EIDE - (Enhanced Integrated Drive Electronics):
• Schnittstelle zum Anschluss von maximal zwei Laufwerken:
• Aktuelle Motherboards bieten zwei EIDE-Schnittstellen direkt auf der Platine
• Der Plattencontroller sitzt auf dem Laufwerk





ATAPI
(Advanced Technology Attachment Packet Interface)
Standard zur Einbindung von „Nicht-Festplatten“ an die (E)IDE- Schnittstelle. Für ein ATAPI-Laufwerk im System braucht man einen „Treiber“.

18.3.5 Physikalischer Zugriff

Unter physikalischem Zugriff wird der unmittelbare Zugriff auf die Platte verstanden. Zum Bearbeiten eines Sektors muss (neben Auswahl der Platte) der Zylinder und die Oberfläche angegeben werden. Innerhalb der so adressierten Spur muss dann der richtige Sektor ausgewählt werden.
Alle andere Zugriffsarten werden vom Betriebssystem emuliert und mit Hilfe des physikalischen Zugriffs abgearbeitet. Innerhalb eines Programms wird diese Zugriffsart nur in Ausnahmefällen verwendet (z.B. zum Beschreiben von Boot-Sektoren).

18.3.5.1 Sequentieller Zugriff

Alle Betriebssysteme kennen den sequentiellen Zugriff auf Dateien. Hier werden hintereinander alle Einheiten (Sektoren, Bytes oder Sätze; je nach Organisation) bearbeitet. Der sequentielle Zugriff ist die mit Abstand häufigste Zugriffsart.

18.3.5.2 Wahlfreier Zugriff (random access)

Beim wahlfreien Zugriff muss die Datei in Sätze fester Länge gegliedert sein. Die einzelnen Sätze können dann unter Angabe der Satznummer direkt angesprochen werden. Mit dieser Zugriffsart kann eine Datei als ein eindimensionales Feld von Sätzen angesehen werden (array of record).

18.3.5.3 Index-sequentieller Zugriff

Unterliegen die Sätze einer Datei einer Ordnung und muss man bestimmte Sätze suchen, so bietet sich eine index-sequentielle Organisation der Datei an.
Jeder Datensatz (Record) hat einen Schlüsselteil, der das Ordnungsmerkmal enthält. Daneben können in einem Record weitere Daten vorhanden sein. Alle Sätze besitzen die gleiche Länge; die Sätze selbst sind nach dem Schlüsselteil sortiert.
Neben diesen Datenrecords enthält die Datei Index-Records. In diesen sind die Schlüssel enthalten und die zugehörigen Verweise auf weitere Sätze (Zeiger). Diese Sätze können wiederum Index-Records oder Daten-Records sein.
Eine index-sequentielle Datei kann man sich also als Baum vorstellen. Die Blätter sind die Datensätze, alle anderen Knoten die Index-Sätze. Der erste Satz einer index-sequentiellen Datei entspricht einer Wurzel.

18.3.5.4 Relationaler Zugriff

Oft kommt es vor, dass Datensätze nach unterschiedlichen Kriterien sortiert oder gesucht werden müssen. Hier scheidet eine index-sequentielle Organisation aus, da nur ein Schlüssel erlaubt ist. Ein Ausweg bietet die Kombination von wahlfreien und index-sequentiellen Zugriff an. In einer Hauptdatei (random-access) sind die Datensätze enthalten. Daneben existieren mehrere Index-Dateien. In jeder Index-Datei ist die Sortierung nach einem Kriterium enthalten (spezifische Schlüssel und Verweise auf Sätze der Hauptdatei).

Jede Index-Datei entsprich wiederum einem Baum. Jeder Baum ist nach einem anderen Kriterium sortiert. Die Blätter der Index-Dateien enthalten in diesem Fall nicht die Datensätze, sondern Zeiger auf die Sätze der Hauptdatei.

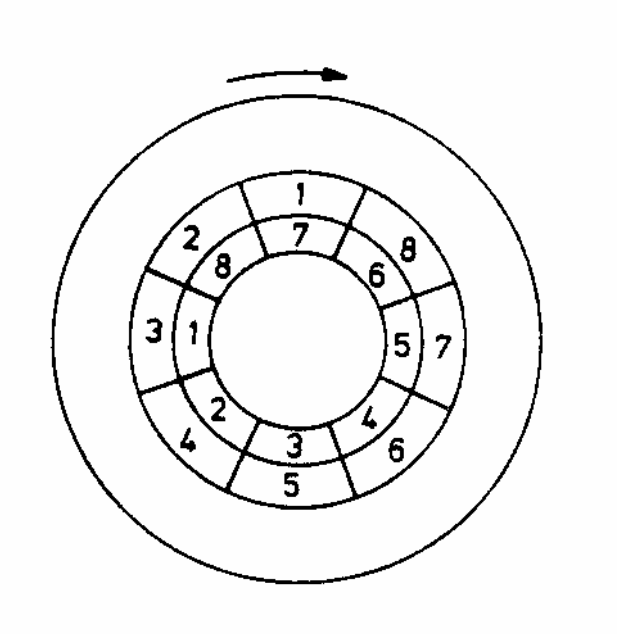
18.3.5.5 Interleaving und Shift-Faktor

Eine der häufigsten Betriebsarten einer Platte ist das sequentielle Lesen von Sektoren. Ein Programm fordert einen Sektor an (der evt. in Sätze zerlegt wird) und bearbeitet ihn. Dann wird der nächste Sektor gelesen. Hier sind zwei Seiten von Bedeutung. Die erste ist die Bebarbeitungszeit des Computers für einen Sektor. Die zweite ist die Zeit, in der sich die Platte vom Ende des gerade gelesenen Sektors zum Anfang des nächsten weiterdreht. Diese Zeit ist verschieden von Null, da bekanntlich Lücken zwischen den Sektoren existieren.
Ist die erste Zeit kleiner als die zweite, so können die Sektoren innerhalb einer Umdrehung gelesen werden. Ist sie größer, so muss zum Lesen des nächsten Sektors auch die nächste Umdrehung abgewartet werden. Dies kostet Zeit.
Abhilfe verspricht hier ein Verfahren, welches Interleaving genannt wird. Die Sektoren einer Datei werden hierbei nicht hintereinander auf die Spur geschrieben, sondern es werden einige Sektoren ausgelassen. Diese Sektoren werden dann entsprechend später verarbeitet.
Hierdurch verkürzt sich die Wartezeit des Computers auf den nächsten Sektor. Man lässt gerade so viele Sektoren verstreichen, dass die Bearbeitungszeit des Computers kleiner wird als die Rotationszeit der ausgelassenen Sektoren.
Der Abstand zwischen zwei in der Datei hintereinander liegenden Sektoren wird Interleaving-Faktor genannt.

Natürlich ist die Bearbeitungszeit von Programm zu Programm verschieden. Man zieht deshalb häufig benutzte Programme (z.B. Copy) als Referenz heran.
Dieser Sektoren-Versatz innerhalb einer Spur kann auf zwei verschiedene Arten erreicht werden. Zum einen kann die Nummerierung der Sektoren auf der Spur selbst versetzt erfolgen. Zum anderen kann das Betriebssystem anhand einer Tabelle eine Übersetzung von logischen Sektor-Nummern in physikalische Sektor-Nummern vornehmen.
Interleaving ist ein sehr oft verwendetes Verfahren, im die effektive Übertragungsrate zu steigern.

Sektor-Interleaving:
Die Sektoren werden verschränkt angelegt
-> Erhöht Durchsatzrate.
Beispiel: Interleavefaktor = 3

Interleaving

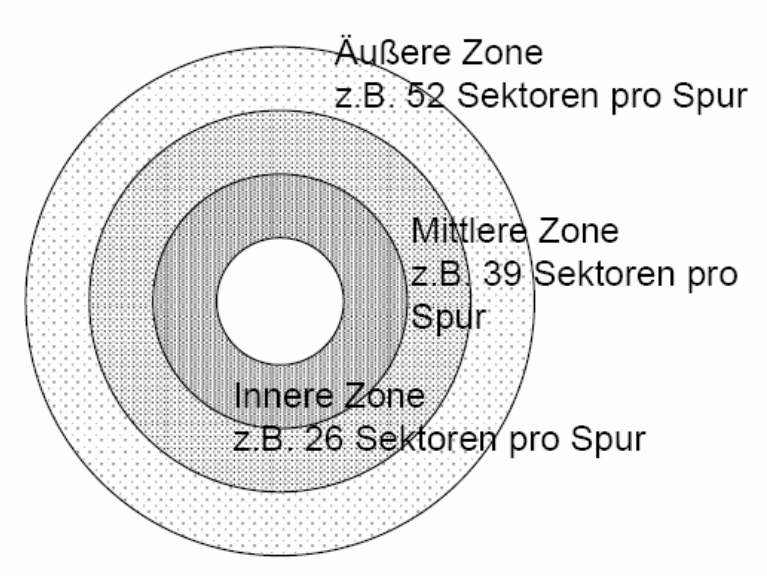


Shift-Faktor

Ein weiterer Punkt beim sequentiellen Lesen von Sektoren ist der Wechsel von einer Spur auf die nächste. Während dieser Zeit ist kein Schreiben oder Lesen möglich. Damit man in diesem Fall keine volle Umdrehung abwarten muss, kann man bei der Nummerierung der Sektoren auf der nächsten Spur mit einer anderen Sektor-Nummer anfangen. Dieser Versatz zwischen zwei Spuren wird Shift-Faktor genannt. Im Gegensatz zum Interleaving ist der Einsatz dieses Verfahrens weniger gebräuchlich.

18.3.5.5.1 Zone-Bit Recording (ZBR):

Erhöht in äußeren Spuren Anzahl Sektoren/Track
-> Kapazitätserhöhung bei gleicher Schreibdichte



18.3.5.6 Fragmentierung

Wenn logisch hintereinander liegende Sektoren auf verschiedenen Bereichen einer Platte liegen, muss beim Bearbeiten dieser Datei der Kopf unnötig oft hin und her bewegt werden. Dies kostet Zeit und macht das Interleaving-Verfahren wirkungslos.
Dieser Effekt wird Fragmentierung genannt und sollte nach Möglichkeit vermieden werden (evtl. durch Reorganisation der Platte).

18.3.5.7 Pufferung

Ein weit verbreitetes Verfahren zur Steigerung des Durchsatzes ist die Pufferung. Hier nimmt ein Teil des Hauptspeichers komplette Bereiche der Platte auf. Dieses Prinzip lässt sich für unterschiedliche Zwecke einsetzen.
Häufig wird z.B. das Directory (oder ein Teil davon) bzw. die FAT gepuffert, da sehr oft darauf zugegriffen wird. Ab dem zweiten Zugriff wird dann nicht mehr von der Platte gelesen, sondern sehr viel schneller direkt aus dem Hauptspeicher.
Bei Benutzer-Programmen werden häufig größere Einheiten (mehrere Sektoren) direkt hintereinander gelesen und zwischengespeichert. Dadurch kann dieses Programm (bei sequenziellem Zugriff) mehrere Sektoren ohne Zeitverlust bearbeiten. Außerdem greift das Programm weniger oft direkt auf die Platte zu, so dass mehr Zeit für Anforderungen anderer Programme zur Verfügung steht. Die Positionierzeit verringert sich im Mittel ebenfalls.

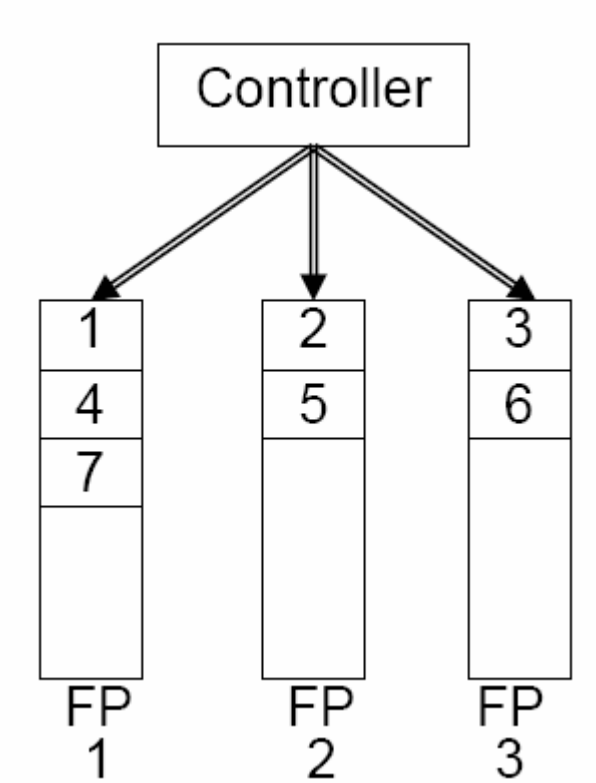
Bei der Pufferung tauscht man also Hauptspeicherplatz gegen Geschwindigkeit ein.

18.4 Raid-Verbünde

RAID = **R**edundant **A**rrays of **I**nexpensive **D**isks
• Konfigurierte Raid-Verbünde sind controller-spezifisch, d.h. nicht portierbar.
• SCSI- oder IDE-Controller werden an den PCI-Bus angeschlossen.
• Datenübertragungsraten aktueller Festplatten (2001):
    • IDE: 30 bis 40 MByte/s
    • SCSI: 40 bis 50 MByte/s
• Festplatten-Cache: 0,5 bis 8 MByte
    • Für Office-Anwendungen ausreichend,
    • Grafik, CAD, Video erfordern größere Dateien.
• Durchsatz des PCI-Busses:
    Bandbreite = (Daten-) Bitbreite x Busfrequenz
    = 32 Bit x 33 MHz
    = 32 Bit x 33 x 106 / s
    = 1056 x 106 Bit/s
    = 132 MByte/s (übliche Angabe: 133 MByte/s)
• Wichtige Raid-Level:
    •Raid 0 Striping
    •Raid 1 Mirroring (Spiegeln)
    •Raid 0+1 Gespiegelte Stripe Sets
    •Raid 5 Stripe Set mit verteilter Parität

18.4.1 Prinzip
18.4.1.1 Raid 0 - Striping

Eigenschaften:
• Ziel: Hoher Datendurchsatz
• Stripe Sets weisen keine Redundanz auf
• Mehrere Festplatten werden zu einem logischen Laufwerk zusammengeschlossen
• Daten werden in Pakete unterteilt und auf alle Festplatten gleichmäßig verteilt
Vorteile:
• Theoretisch addieren sich die Datendurchsätze der einzelnen Festplatten
Durchsatz(Raid) = Durchsatz(Festplatte) x Anzahl Festplatten
praktisch setzen Verwaltungsoverhead und Leistungsgrenzen der Controller (IDE, SCSI) und des PCI-Bus Grenzen
• Speicherkapazität der Festplatten addiert sich



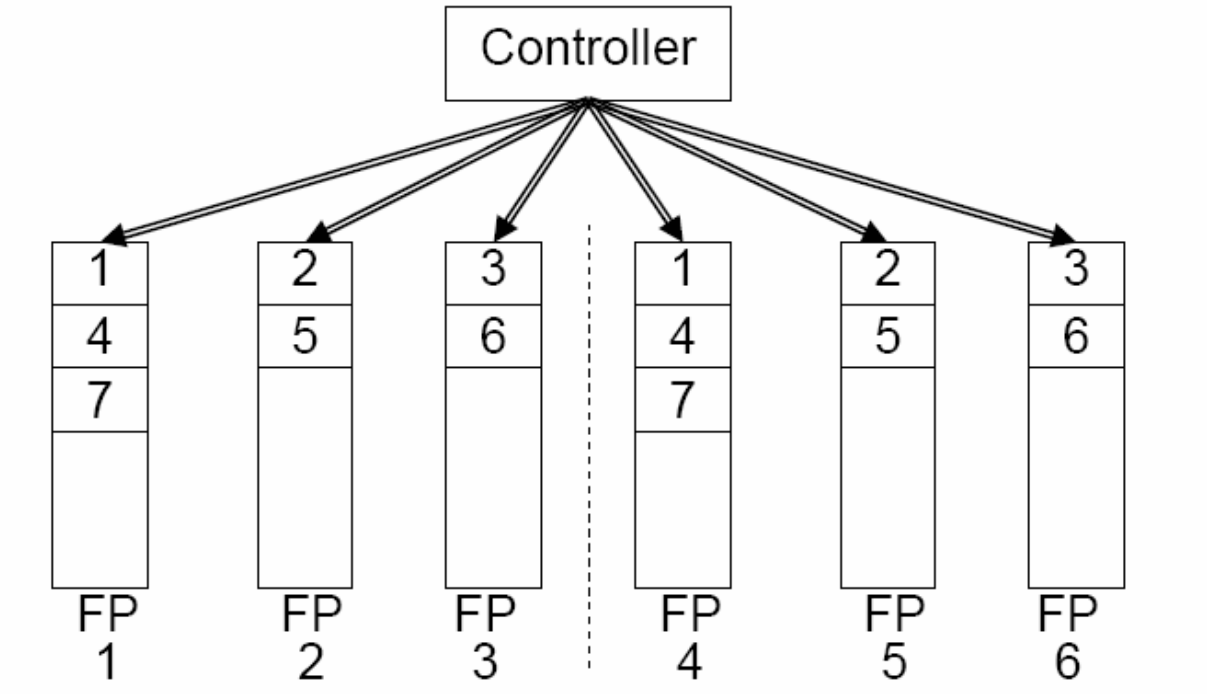
Nachteile:
• Langsamste Festplatte im Set bestimmt die Gesamtleistung (gleiche Platten verwenden)
• Verringerung der Datensicherheit – Defekt einer Festplatte führt zu vollständigem Datenverlust
MTBF(Raid) = MTBF(Festplatte) / Anzahl Festplatten
MTBF = Mean Time between Failures (herstellerspezifische Angaben) Beispiel:
MTBF von 50.000 Stunden (fast 6 Jahre) je Festplatte entspricht mit 4 Festplatten
12.500 Stunden (1,5 Jahre) des Sets

18.4.1.2 Raid 1 – Mirroring (Spiegelung)

Eigenschaften:
• Jede Festplatte wird im Verbund gespiegelt
• Raid 1 – Systeme bestehen aus (1+1), (2+2), (3+3), ...Festplatten
• Schreiben erfolgt synchron auf beide Festplatten
• Lesen erfolgt von einer Festplatte, Load Balancing: Lesen jeweils von der Festplatte, die gerade nichts zu tun hat
Vorteile:
• Sicherheit
MTBF(Raid) = (MTBF(Festplatte) x 2) / 2 x MTTR
MTTR = Mean Time to Repair (Zeit bis das defekte Laufwerk repariert ist)
Nachteile:
• Es steht nur die Kapazität einer Festplatte zur Verfügung (die der kleinsten im Verbund)
• Datendurchsatz im Verbund wird durch die langsamste Festplatte bestimmt

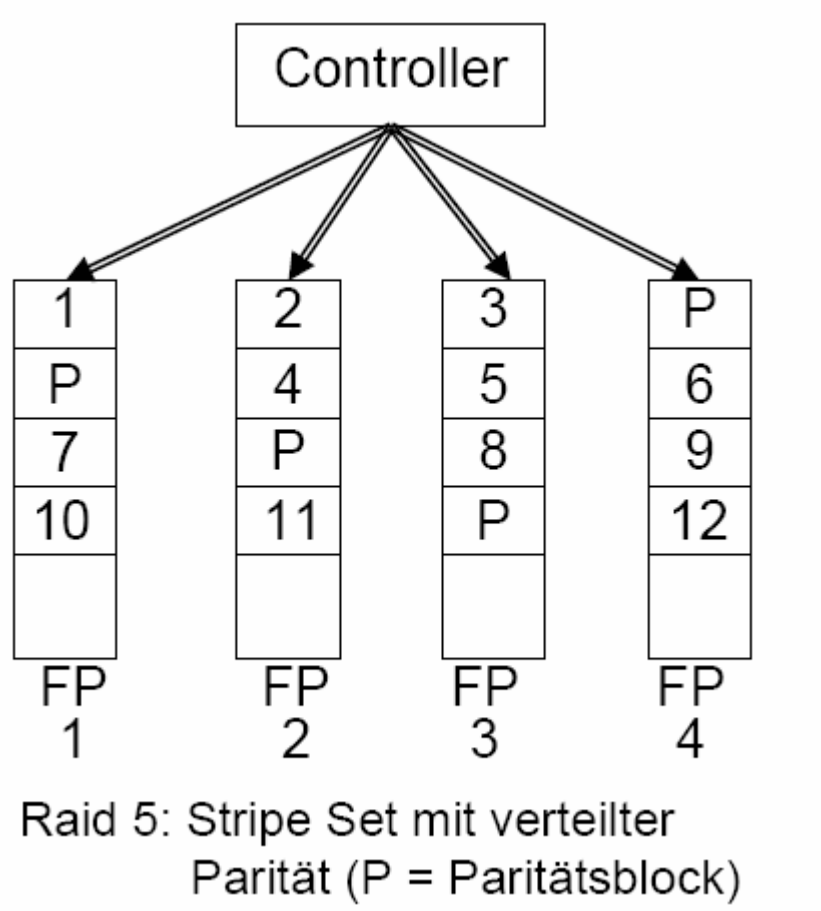
18.4.1.3 Raid 0+1 (Raid 10)

Eigenschaften:
• Raid 0+1 wird oft Raid 10 genannt
• Ziel: Verbinden der positiven Eigenschaften von Raid 0 und Raid 1
• Es werden 2 Stripe Sets gespiegelt
Vorteile:
• Hohe Datensicherheit
• Hoher Datendurchsatz
Nachteile:
• Mindestens 4 Festplatten sind erforderlich
• Bei z.B. 4 Festplatten steht nur die Kapazität von 2 Festplatten zur Verfügung



18.4.1.4 Raid 5

Eigenschaften:
• Raid 5 wird meist in Servern in Verbindung mit SCSI-Festplatten eingesetzt
• Es werden Paritätsdaten gespeichert, die den Ausfall einer Platte kompensieren können
• Unterschied zu Raid 3 und 4: Paritätsdaten werden wie Daten auf die verschiedenen Festplatten verteilt
• Jede Platte enthält sowohl Nutzdaten als auch Paritätsdaten
Vorteile:
• Hoher Datendurchsatz (fast wie Stripe Set)
• Hohe Sicherheit (fast wie Mirroring)
• Relativ geringer Plattenoverhead
Nachteile:
• Es sind mindestens drei Festplatten erforderlich
• Teure Controller (bis zu mehreren K€)



Raid 5: Stripe Set mit verteilter Parität (P = Paritätsblock)

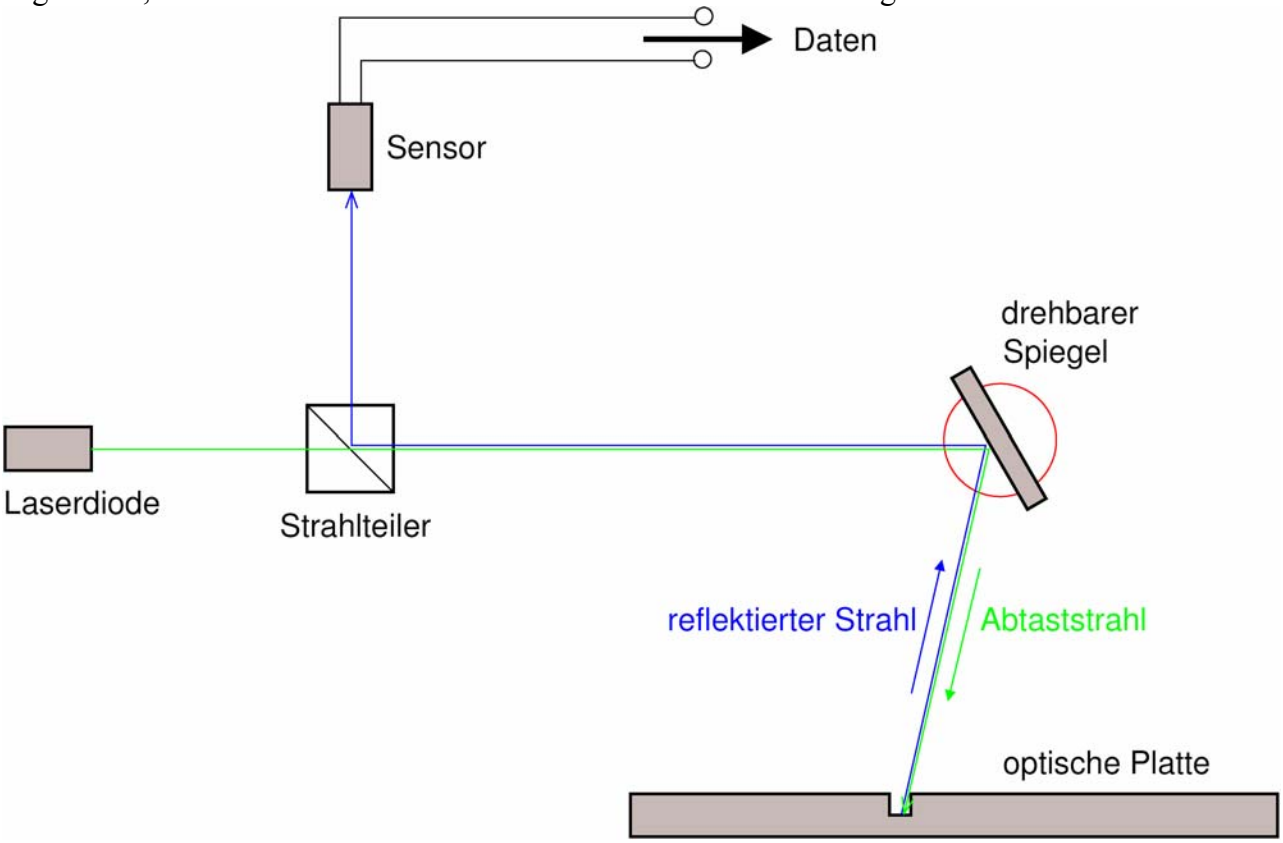
18.5 Multimedia-Anforderungen an Festplatten

Anforderungen:
• High Quality Video
(30 Bilder/s) x (640 x 480 Pixel) x (24-Bit Farben / Pixel) = 221 Mbit/s (28 MByte/s)
->entspricht etwa der mittleren Transferrate aktueller Festplatten
• Reduced Quality Video
(15 Bilder/sec) x (320 x 240 Pixel) (16-bit Farben / Pixel) = 18 MBit/s (2,2 MByte/s)
• High Quality Audio
(44.100 Audio Samples / s) x (16-bit je Sample) x (2 Kanäle für Stereo) = 1.4 MBit/s
->Datenkompression hat großen Einfluss auf die Anforderungen an die Hardware!
Trends der Festplatten Performance
• Die Speicherkapazität erhöht sich um ca. 60% pro Jahr (Verdopplung alle1,5 Jahre).
• Die Transferrate wächst ca. 40% pro Jahr (Verdopplung alle 2 Jahre).
• Rotationen der Platten und die Sektor-Suchzeiten verbessern sich um ca. 8% pro Jahr (also etwa x 0,5 in 10 Jahren).
• Die Kosten pro MByte fallen jedes Jahr auf etwa 60%.
• Laufwerke benötigen immer weniger Chips und haben eine höhere Bitdichte.

18.6Optische Aufzeichnung

18.6.1 Prinzip

Licht wird in Abhängigkeit von der Oberfläche der reflektierenden Schicht bei der Reflektion unterschiedlich stark gestreut. Zur Speicherung von Daten wird die plane Oberfläche eines optischen Speichermediums manipuliert. Auf der Scheibe werden kleine Vertiefungen angebracht, wodurch der reflektierte Strahl an diesen Stellen stärker gestreut wird.



Prinzip des Aufbaus zum Lesen von Daten einer CD-ROM

In obiger Abbildung ist das Prinzip für das Auslesen von Daten von einer CD-ROM dargestellt. Der Strahl eines kleinen Halbleiterlasers wird über einen bewegbaren Spiegel auf das Medium geschickt. Der reflektierte Strahl wird mit Hilfe eines Sensors analysiert. Das Medium reflektiert im Grundzustand den Laserstrahl ohne starke Streuungen. Dies wird durch eine Verspiegelung der Oberfläche erreicht. Mit einer Modifikation der Oberfläche durch kleine Löcher oder Verätzungen wird an den betreffenden Stellen eine deutliche stärkere Streuung des einfallenden Strahls erreicht. Dieses Prinzip korrespondiert mit der Ausrichtung der Weiß'schen Bezirke bei den diskutierten magnetischen Laufwerken.
Bei der CD-ROM werden üblicherweise kleine Vertiefungen, die Pits genannt werden, in einer ansonsten sehr gut reflektierenden Schicht eingebracht. Jeder Übergang von einem Pit zur unveränderten Oberfläche, die als Land bezeichnet wird, wird als eine Eins interpretiert. Auch der Übergang von Land zu einem Pit wird als Eins interpretiert. Eine Strecke von 300 nm ohne Veränderung wird hingegen als Null interpretiert. Durch diese Weise der Datenspeicherung kann eine Folge von Einsen nicht gespeichert werden. Daher muss eine Codierung gefunden werden, die nur einzelne Einsen und Folgen von Nullen enthält. Hierfür werden 8-Bit-Daten in 14-Bit-Daten umgesetzt, die den Anforderungen der optischen Speicherung genügen. Bei der industriellen Herstellung von CD-ROM werden die Vertiefungen mit Hilfe einer eines Stempels der Negativabdrucks in die Scheiben gepresst.

Zur Datencodierung muss die Reflektionseigenschaft des Mediums geändert werden. Das Einbringen von Vertiefungen ist hierfür nur eine Möglichkeit. Die Veränderung der Reflektionseigenschaften kann auch auf andere Weise erreicht werden. So kann die Oberfläche des Trägers unterschiedlich magnetisiert werden oder zwei verschieden Phasen (kristallin / amorph) haben.
Bisher wurde nur das Lesen einer CD-ROM besprochen. Im folgenden wird nun noch das Prinzip der CD-Recordable bzw. der CD-ReWritable dargestellt. Bei diesen Geräten wird die unterschiedliche Lichtdurchlässigkeit in Abhängigkeit von der Phase des Aufzeichnungsmaterials ausgenutzt.
Zur Realisierung von CD-Recordables (CD-R) wird eine organische Schicht zur Veränderung der Reflektionseigenschaften der reflektierenden Schicht eingesetzt. Der abtastende Laserstrahl wird in Abhängigkeit von der Beschaffenheit der organischen Schicht reflektiert. Zur Reflektion wird häufig eine Goldschicht eingesetzt. Die organische Schicht ist so beschaffen, dass sich bei einer Erhitzung durch den Schreiblaser des CD-R-Laufwerks eine winzige lokalisierte Blase bildet. Diese Blase vermindert die Reflektion des Abtaststrahls an dieser Stelle. Die Blasenbildung ist irreversibel, so dass die geschriebenen Daten nicht gelöscht werden können. Das Lesen der Daten erfolgt nach dem selben Prinzip wie im Fall der CD-ROM. Es kann sogar derselbe Laer zum Lesen der Daten verwendet werden, so dass eine CD-R in fast allen CD-ROM Laufwerken ausgelesen werden kann. Das Lesen erfolgt mit einer deutlich geringeren Energie als das Schreiben der Daten, so dass es nicht zu unbeabsichtigter Blasenbildung kommen kann.
Für die CD-ReWritable (CD-RW) wird als Trägermaterial Polykarbonat verwendet. Die Reflektionsschicht besteht aus Aluminium. Hinzu kommt eine sogenannte Phasenwechselschicht. Diese Technologie wird daher als Phase-Change-Technology bzw. Phasenwechseltechnologie bezeichnet. In der Phasenwechselschicht wird eine Veränderung der optischen Eigenschaften durch eine Änderung der Kristallstruktur des Materials erreicht. Für das Setzen eines Bits wird ein kleiner Bereich lokal ($< 1\mu m^2$) mit Hilfe eines fokussierten Laserstrahls auf über 600 Grad Celsius erhitzt. Damit wird die kristalline Struktur in diesem Gebiet geschmolzen. Die Schmelze wird anschließend extrem schnell abgekühlt, so dass eine kristalline Strukturbildung verhindert wird. Die amorphe Struktur der Schmelze bleibt erhalten. Durch die amorphe Struktur wird die Reflektion vermindert. Dies entspricht den Pits im Fall der CD-ROM. Die kristalline Struktur der Bereiche kann wieder hergestellt werden indem die Bereiche mit Hilfe des Laserstrahls auf eine mittlere Temperatur erwärmt werden. Die so zugeführte Energie genügt zur Ausrichtung der Atome einer kristallinen Struktur. Der stark reflektierende Zustand ist wieder hergestellt. Die gespeicherten Daten sind gelöscht. Zum Lesen der Daten wird das schon oben erläuterte Prinzip der Messung der reflektierten Strahlung angewandt. Das Lesen erfolgt mit einer Energie die nochmals deutlich geringer ist als die Energie zum Löschen der Daten. Der Kontrast zwischen amorphen und kristallinen Bereichen ist mit 75% sehr groß.
Die Digital Versatile Disk (DVD)-ROM arbeitet mit dem selben Prinzip wie die CD-ROM. Zur Speicherung der erheblich höheren Datenmenge müssen aber diverse Änderungen durchgeführt werden. Pits und Lands sowie deren Abstände wurden verkleinert. Der Laser muss daher erheblich exakter fokussiert werden. Aus diesem Grund werden für DVD-Laufwerke als Leselaser im roten Bereich verwendet (635nm oder 650nm). Für CD-ROM werden Laser im infraroten Bereich verwendet (780nm). Die Adressierungs- und Fehlerkorrekturmechanismen wurden verbessert. Auf einer Informationsschicht können auf diese Weise maximal 4.7 GByte Daten gespeichert werden. Darüber hinaus können auf einer DVD zwei Informationsschichten übereinander angeordnet werden. Dies stellt natürlich erhöhte Anforderungen an die Beschichtung der Medien, da die Daten beim Lesen den beiden Schichten eindeutig zugeordnet werden müssen. Zusätzlich können DVDs doppelseitig realisiert werden. Mit diesen Maßnahmen können maximal ca. 17 GByte Daten auf einer

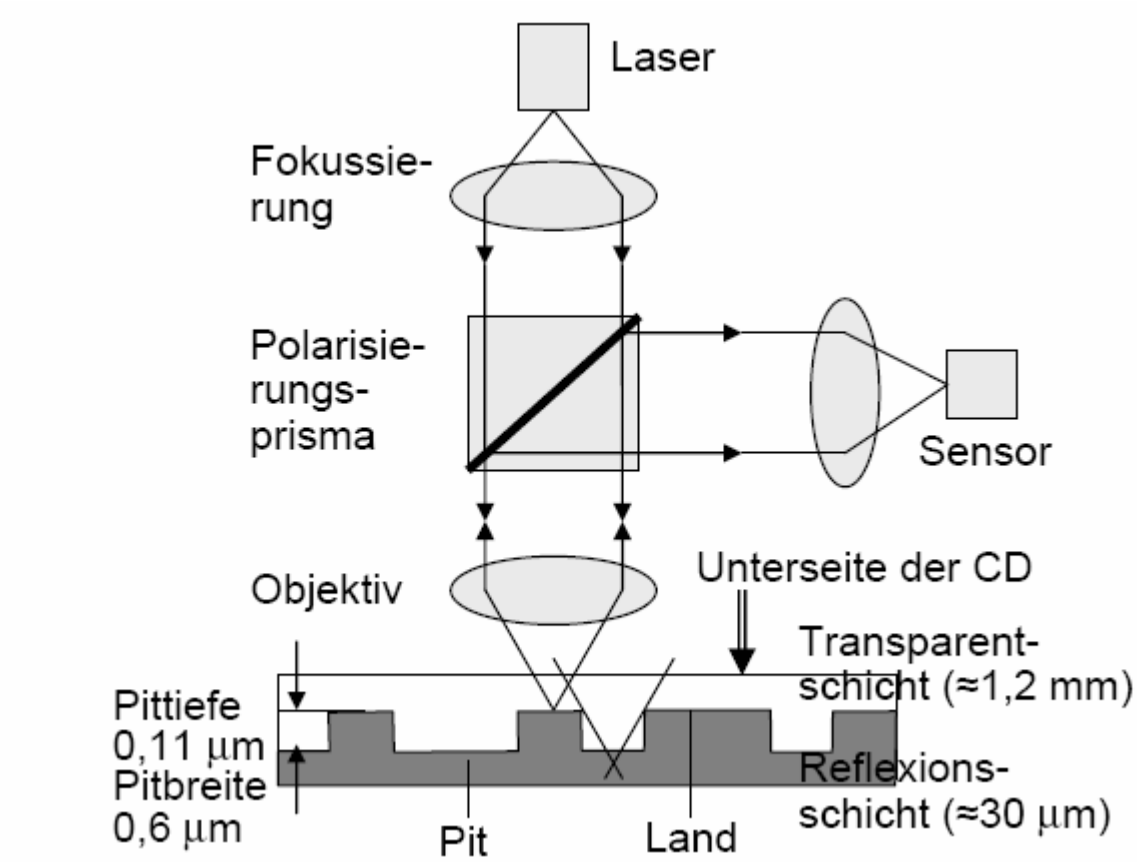
DVD gespeichert werden, wenn sie auf jeder Seite zwei Informationsschichten hat. DVD-RW werden mit der Phasenwechseltechnologie realisiert, die auch bei CD-RWs verwendet wird und oben dargestellt wurde.

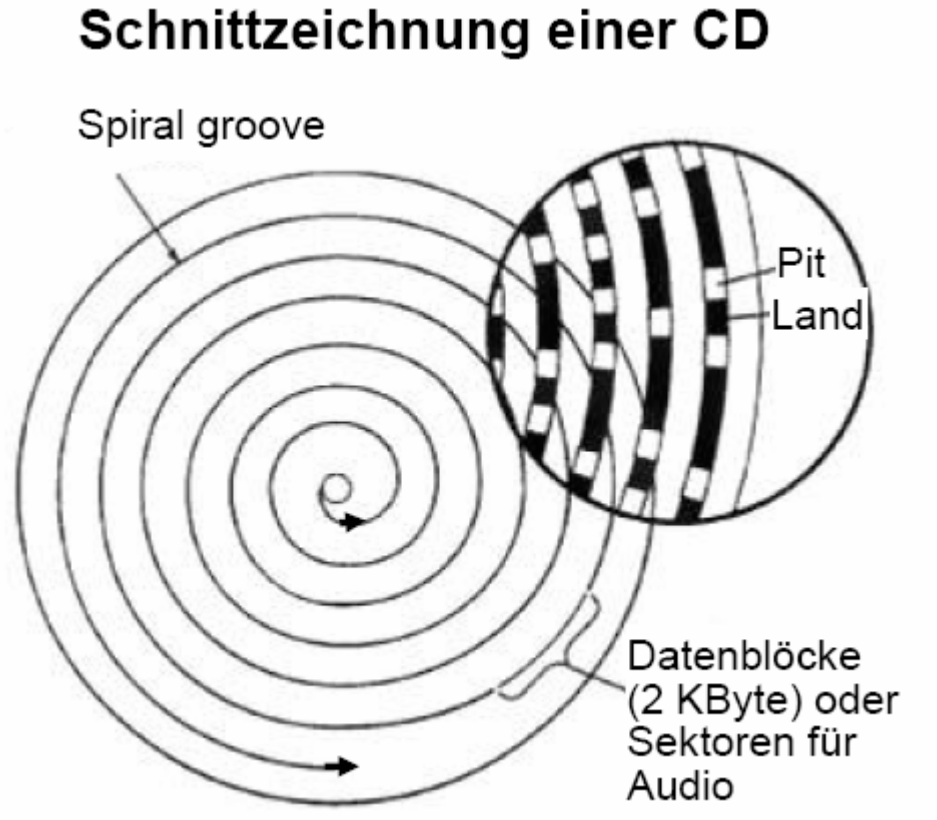
18.6.2 Magnetooptische Laufwerke

Bei magnetooptischen Laufwerken (MO) die Beeinflussung der Polarisation elektromagnetischer Wellen durch Magnetfelder ausgenutzt. Die Polarisierung von linear polarisiertem Licht kann mit Hilfe von Polarisationsfiltern ermittelt werden. Erfasst der hinter dem Polarisationsfilter angebrachte Detektor Licht, so ist das Licht entsprechend der Richtung des Filters polarisiert. Lässt der Polarisationsfilter hingegen kein Licht durch, so ist dieses senkrecht zur Richtung des Filters polarisiert. Die Polarisationsrichtung des Abtastsignals wird durch die Ausrichtung der Elementarmagnete in dem Bereich bestimmt, der aktuell abgetastet wird.
Die magnetooptische Schicht des Speichermediums ist häufig eine Terbium-Ferrit-Kobald-Legierung. Zum Speichern eines Datums wird der entsprechende Bereich bis zum Curie-Punkt des Materials, der bei ca. 200 Grad Celsius liegt, erhitzt. Hierdurch wird die Magnetisierung der ferromagnetischen Schicht vollständig aufgehoben. Gleichzeitig wird ein Magnetfeld aufgebaut, das die Elementarmagnete entsprechend des zu schreibenden Datums ausrichtet. Direkt anschlie0ßend wird das Material abgekühlt, so dass die Ausrichtung der Magnetisierung nicht mehr durch Störungen aufgehoben werden kann. Das Prinzip entspricht der oben dargestellten magnetischen Speicherung von Daten, die ebenfalls auf einer magnetischen Ausrichtung beruht. Allerdings wird bei der magnetooptischen Speicherung ein Ferromagnetikum gewählt, dessen Magnetisierung unterhalb der Curie-Temperatur nicht verändert werden kann. Zum Lesen der Daten wird das Medium mit einem Leselaser abgetastet, der das Medium nur geringfügig erwärmt. Das Polarisationssystem des Laufwerks detektier die Polarisationsrichtung des reflektierten Strahls. Aus der Drehrichtung kann auf das gespeicherte Datum zurück geschlossen werden. Die magnetooptische Speicherung gilt als die sicherste Variante der Datenspeicherung, da zur Löschung der Daten gleichzeitig eine hohe Temperatur und ein Magnetfeld benötigt wird.

18.7 CD-ROM

Modulation des reflektierten Laserlichts
• Pittiefe von 0,11 µm entspricht ¼ der Lichtwellenlänge in Kunststoff
• Phasenverschiebung von 2 x ¼ λ = ½ λ (Rückweg mitgerechnet)
• Interferenz: Pit: Abschwächung, Land: Verstärkung





• Es gibt nur eine Spur, die als Spirale von Innen nach Außen führt.
Ziel: Unterstützung einer kontinuierlichen Datenrate.
• Spurbreite = 0,6 µm, Abstand zweier Windungen = 1,6 µm

-> 650 MB-CD (74 Min.) ca. 20.000 Windungen (LP ≈ 850 Windungen)

18.7.1 Prinzip
18.7.1.1 Formate

Audio-CD (Red Book):
• Lead-In:
Name der CD, Autor, Verleger, Datum, Inhaltsverzeichnis mit Angaben des Beginns der logischen Tracks
• Datenbereich: Bis zu 99 Tracks mit Nutzdaten
• Lead-Out: Kennzeichnung des CD-Endes (90 Sekunden Stille)
• Adressierung der Daten: Minuten:Sekunden:Sektor (MM:SS:DD)
• Sektoraufbau: 2352 Byte Nutzdaten je Sektor
• Datenrate: (75 Sektoren/s) x 2352 = 175.400 Byte/s;
-> entspricht 44,1 kHz in 16 Bit Stereo:
44,1 kHz x (2 Stereokanäle) x (16 Bit = 2 Byte) = 176.400 Byte/s
• Spielzeit: 330.000 Sektoren ergeben etwa 74 Minuten

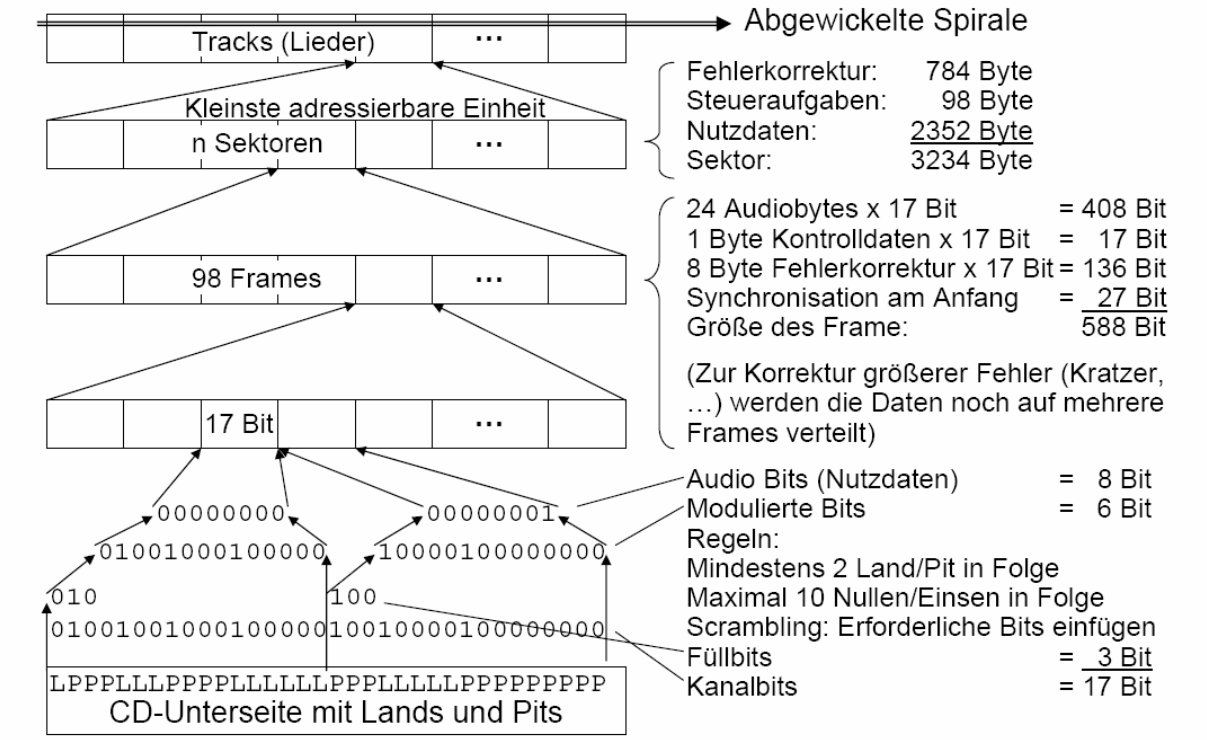
CD-ROM (Yellow Book):
• Lead-In: Zusätzlich Inhaltsverzeichnis
(TOC = Table Of Contents) und Name der Startapplikation

CD-ROM Mode 1:
• Computerdaten
• Layered Error Correction: Fehlerrate = 10^{-12} (vergleichbar der von Festplatten)
• Synchronisation: 12 Byte, Sektorheader: 4 Byte
• 2048 Byte Nutzdaten je Sektor

CD-ROM Mode 2:
• Komprimierte Audio-, Grafik-, Bilddaten
• Keine zusätzliche Fehlerkorrektur
• Synchronisation: 12 Byte, Sektorheader: 4 Byte
• 2336 Byte Nutzdaten je Sektor

Anmerkung:
Es gibt noch weitere Spezifikationen, z.B. „Green Book“

18.7.1.2 Logische Struktur Audio-CD / Daten-CD



18.7.1.3 Erstellen von CD

Ein Block (entspricht bei Festplatten einem Sektor) besteht aus 98 Frames (wie Audio-CD). Blöcke sind 2.352 Byte lang. Man unterscheidet drei Arten von Blöcken:

CD-ROM Mode 0

Sync 12	Header 4	NULL 2.048	EDC 4	Leer 8	ECC 276

Alle Nutzdaten auf 0, dient zur Trennung zwischen Bereichen

CD-ROM Mode 1

Sync 12	Header 4	Nutzdaten 2.048	EDC 4	Leer 8	ECC 276

Ablage von Computer-Daten:
- 12 Byte Synchronisation - Anfangskennung des Blocks
- 4 Byte Header: Nummer des Blocks, Minuten, Sekunden und den Modus
- 2048 Byte Nutzdaten
- 4 Byte zur Fehlererkennung (Error Detection Code: EDC)
- 8 Byte ungenutzt
- 276 Byte zur Fehlerbehebung (Errror Correction Code: ECC)

Bei einer Spieldauer von 74 Minuten gibt es 333.000 Blöcke (650 Mbyte)

CD-ROM Mode 2

Sync	Header	Nutzdaten
12	4	2.336

Ablage von Daten ohne Fehlerkorrektur (z.B. komprimierte Videos)
333.000 Blöcke ergeben 740 Mbyte

18.7.1.4 Erstellen von CD

Industrielle Fertigung von CD-ROM

1. Premastering
   - Berechnung der EDC/ECC-Bereiche
   - Einfügen von Synchronsationsbits
   - Erstellung von Verzeichnissen
2. Mastering
   - Nach Premaster-Image wird ein Glasmaster mit Laser belichtet
   - Belichtete Stellen (spätere Pits) werden entwickelt und ausgewaschen
   - Bedampfen mit Silberschicht und Qualitätsprüfung
3. Herstellung der Matrizen (Stamper, Negative des Glasmasters)
4. Pressen der CD mit dem Stamper
5. Bedrucken und Verpacken

CD Recordable (CD-R)

- Werden industriell durch „Pressen" mit Matrizen gefertigt, danach können sie nur noch einmal beschrieben werden.

Brennen:

- Erfolgt innerhalb einer vorgravierten Spur (Pre Groove, Helix Rille) durch Erhitzen auf über 250 °C.
- Der Laser verändert das Reflexionsverhalten einer Absorptionsschicht, die sich zwischen Substrat und Reflexionsschicht befindet.
- Danach wird Licht bei einem Pit (gebrannte Stelle) schlecht und bei einem Land gut reflektiert.
- Die Vorgravierung wird während des Brennens abgetastet (Positionierung auf Spurmitte und Informationen über das zeitliche Verhalten der CD).

Lesen:

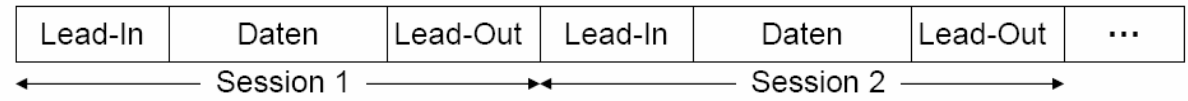
- Bei CD-R gibt es keine Höhen und Tiefen, sondern nur unterschiedliches Reflektionsverhalten.

18.7.1.5 Wiederbeschreibbare CD

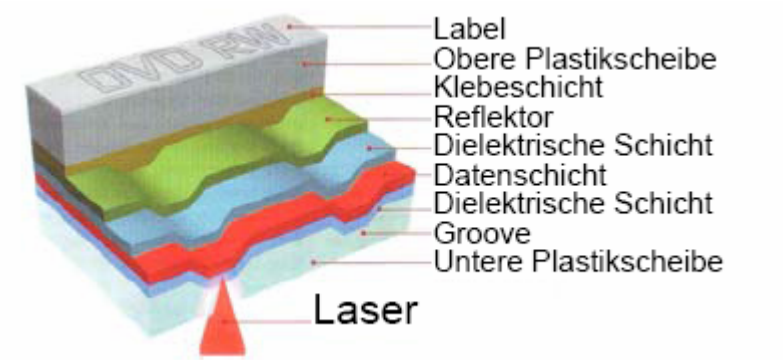
CD - Rewritable (CD-RW)

• Theoretisch bis zu 1000x beschreibbar
• Reflektierende Schicht ist eine Silber-Indium-Antimonium-Tellurium-Legierung, die folgende Zustände annehmen kann:
    • Kristallin und gut reflektierend (ursprünglicher Zustand, Land)
    • Amorph und schlecht reflektierend
• Diese reflektierende Schicht wird mit einem Laser auf 500-700°C erwärmt und wird flüssig, wobei der kristalline Zustand in den amorphen übergeht (Pit).
• Wird diese Stelle später mit 200°C erwärmt, so bleibt die Stelle fest, geht aber in die kristalline Struktur wieder zurück (gute Reflektion, Land).

Sitzungen (Sessions)
• CD-R und CD-RW können in mehreren Schritten beschrieben werden.
• Eine Session ist eine Wiederholung der globale Struktur der CD:
    ->Lead-In, Datenbereich und Lead-Out.
• Theoretisch können 99 Sessions geschrieben werden.

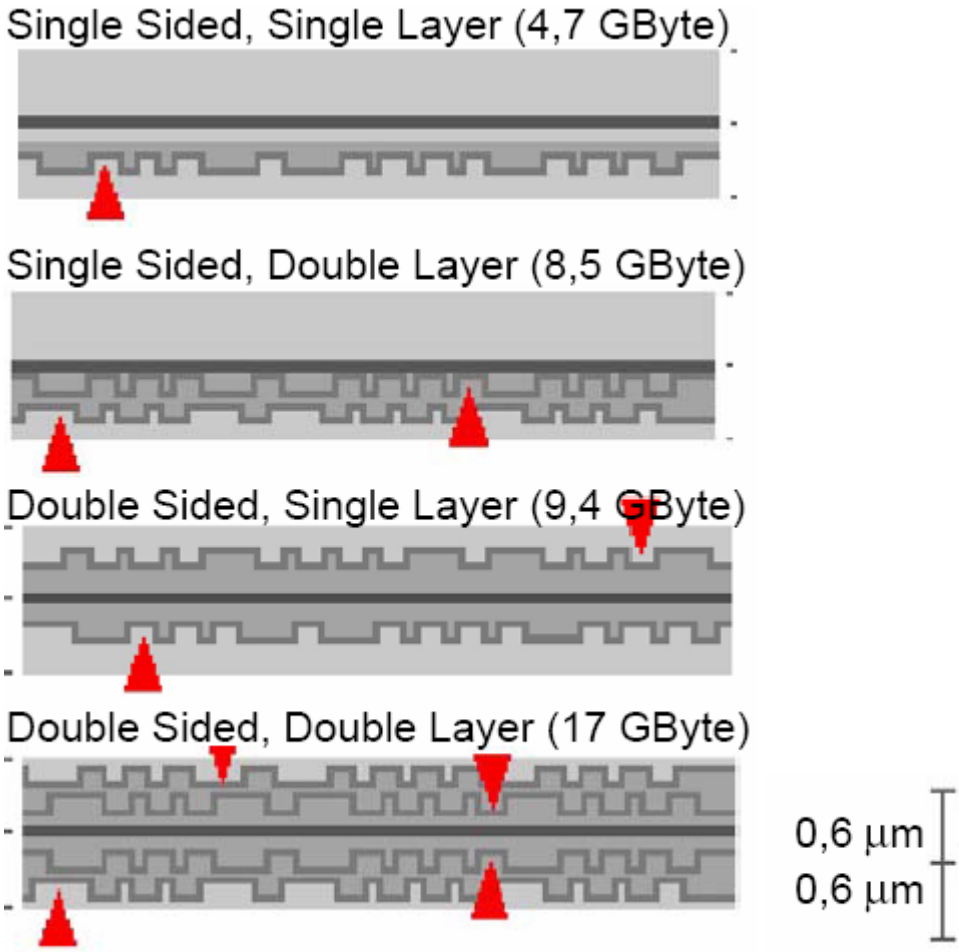


18.8 Digital Versatile Disk – DVD



Frühere Bezeichnung: Digital Video Disk (versatile = vielseitig)
• Funktionsprinzipien sehr ähnlich der CD-ROM
• Höhere Datendichte als CD-ROM (kleinere Pits, höhere Trackdichte, vergrößerter Datenbereich)
• Effizientere Bitcodierung: für 1 Byte jetzt 16 Bit (statt 17 bit: 14+3 bei CD)
• Lead-In und Lead-Out etwas kleiner als bei CD
• Bessere Fokussierung des Laser, verbessertes optisches System
• Licht mit kürzerer Wellenlänge
• Verringerung Mindestlänge Pit (0,9 auf 0,4 µm)
• Verringerung des Spurabstandes (1,6 auf 0,7µm) ->dadurch 4,7 GB pro Informationsschicht möglich
• Sektorgröße einheitlich 2064 Byte (2048 Byte)
• Speicherung in standardisiertem Datei-Format (Universal Disc Format)

Mehrschichtspeicherung:
Zwei Schichten pro Seite sowie Vorder- und Rückseite
-> nahezu 4-fache Kapazität: 17 GB



19 Automaten

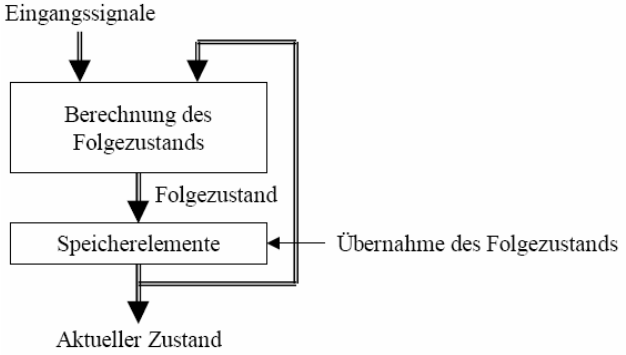
Endliche Automaten beschreiben auf abstrakte Weise das Verhalten von Rechnerhardware, z.B. von Schaltwerken. Ein oft zitiertes - allerdings sehr einfaches - Beispiel dafür ist eine Ampelsteuerung.
Sie kann Eingaben (Inputs) in eine Ausgabe (Output) umwandeln und Daten speichern.
Abstrakt gesehen stellt sie einen endlichen Automaten dar mit Eingaben, Ausgaben und Zuständen. Der Begriff Zustand steht für die jeweilig gespeicherte Information.
Eingaben: Timer input, Knopfdruck Fußgänger, Signal Induktionsschleife...

Ausgaben: Lichtsignale
Zustände: Nord/Süd fahrt, halten, stehen...., Notbetrieb

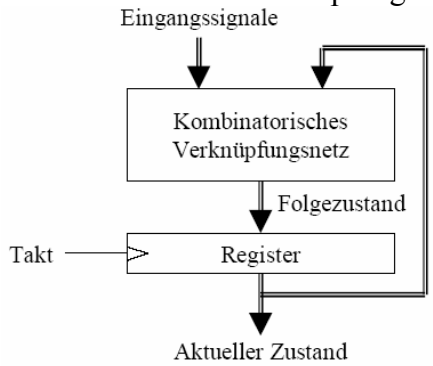
19.1 Endliche Automaten

Struktur:
Bei synchron rückgekoppelten Schaltungen wird der aktuelle Zustand der Schaltung in einem Speicher gehalten.
Aus dem Zustandsvektor und den Eingangssignalen der Schaltung wird ein neuer Folgezustand berechnet und an den Eingang des Speichers angelegt. Synchron wird dann der

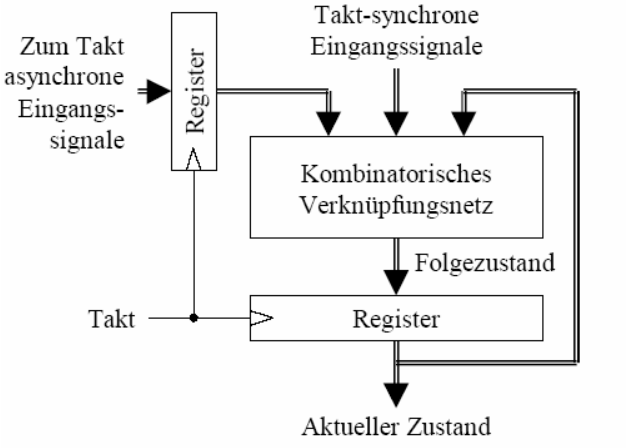
Folgezustand in den Speicher übernommen und damit als aktueller Zustandsvektor aktiviert. Dieses Vorgehen lässt sich in folgendem Blockschaltbild darstellen:



Als Speicherelement eignen sich die in Abschnitt 7.1 vorgestellten Register. Die Übernahme des Folgezustands in den aktuellen Zustand erfolgt dann mit der aktiven Flanke des Takts. Im Zeitraum zwischen zwei aktiven Flanken wird aus dem aktuellen Zustand und den Eingangssignalen der nächste Folgezustand über ein kombinatorisches Verknüpfungsnetz ermittelt:



Zu beachten ist, daß sich die Eingangssignale nur in bestimmten Zeitintervallen ändern dürfen. Es muß immer gewährleistet sein, daß die kombinatorische Verknüpfung zur Berechnung des Folgezustands im Bereich der aktiven Taktflanke in einem eingeschwungenen, stabilen Zustand ist. Nur dann kann ein korrekter Übergang in den Folgezustand gewährleistet werden. Sind Eingangssignale im Automaten zu verarbeiten, die sich zu jedem beliebigen Zeitpunkt ändern können, müssen diese mit einem geeigneten Flip-Flop auf den Takt des Automaten synchronisiert werden. Nachfolgende Abbildung zeigt die Behandlung dieser zum Takt asynchroner Signale:



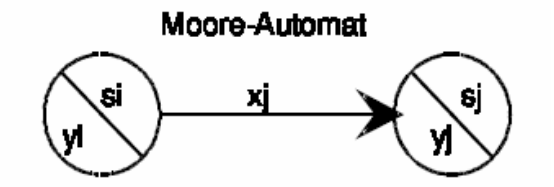
Definition: Ein endlicher Automat ist ein Fünftupel A = (X, Y, S, f, g), wobei X ein endliches nichtleeres Eingabealphabet, Y ein endliches nichtleeres Ausgabealphabet, S eine endliche

nichtleere Menge von Zuständen ist; f: X × S -> S ist die Zustands(überführungs)funktion, g: X × S -> Y ist die Ausgabefunktion.

Ein Automat startet in einem Anfangszustand. Durch Eingangssignale wechselt der Automat gemäß seiner Übergangsfunktion in einen Folgezustand, dabei entstehen neue Ausgangssignale. Wir nehmen hier der Einfachheit halber an, daß sich der Zustand nur bei Vorliegen eines Taktimpulses ändern kann, synchroner Automat.

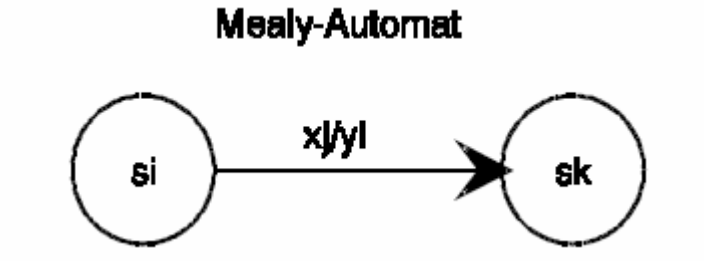
19.1.1 Moore-Automat

Sind die Ausgangssignale nur abhängig vom Zustand, so spricht man vom Moore-Automat.



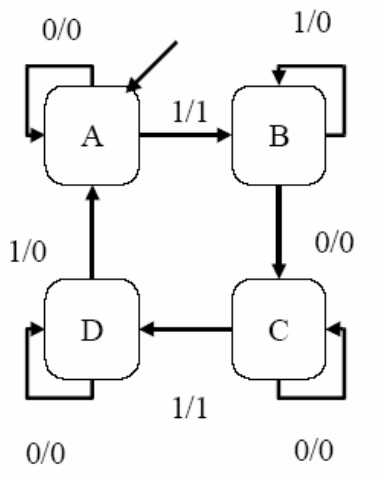
19.1.2 Mealy Automat

Mealy-Automaten wechseln schon mit einem veränderten Eingangssignal ein entsprechendes Ausgangssignal, nicht erst beim Zustandswechsel im nächsten Takt.

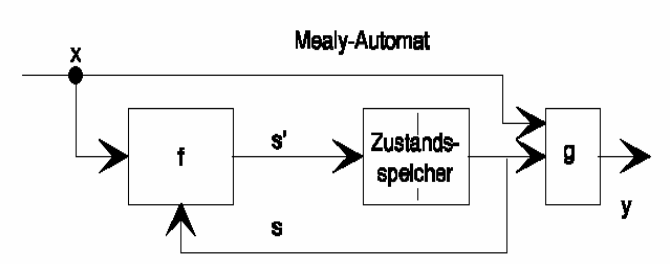


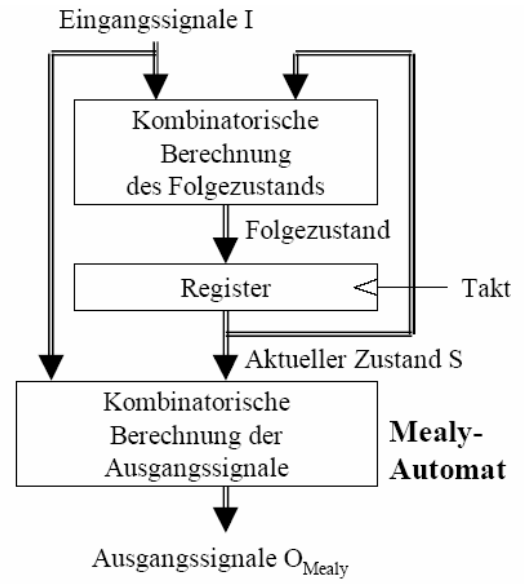
19.1.3 Zustandsgraph

Ein Zustandsgraph besteht aus Knoten und gerichteten Kanten. Die Knoten (Kreise) beschreiben die Zustände. Die Kanten stellen in Abhängigkeit vom anliegenden Eingabezeichen die Übergänge zwischen dem gegenwärtigen und dem nächsten Zustand her Der Zustandsgraph eines Automaten wird gelegentlich mit dem Automaten selbst identifiziert.
Vor dem ’/’ steht die Eingangsbedingung, unter der der Ausgangszustand verlassen wird, hinter dem ’/’ steht beim Mealy-Automat das Ausgangssignal.
Beispiel Mealy Automat A1:

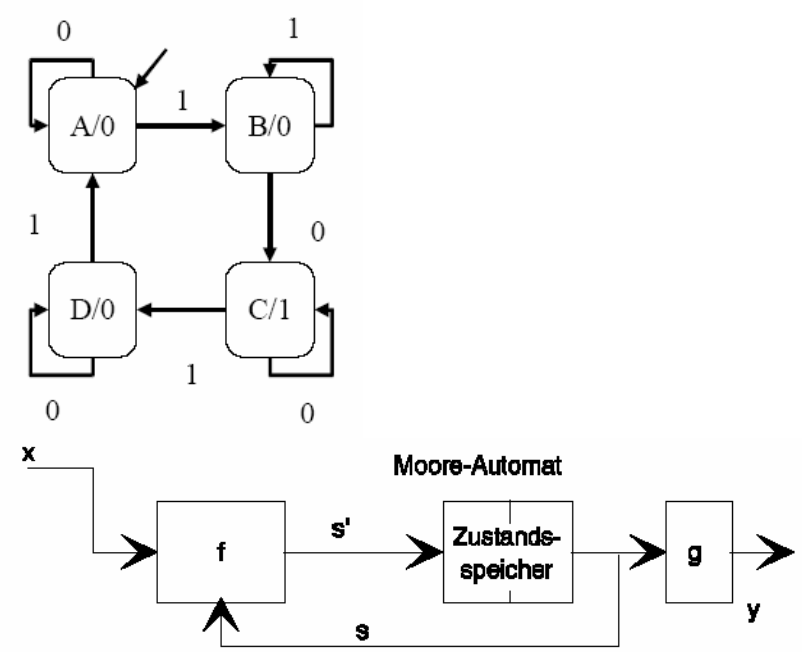


A1 = ({0,1}, {0,1}, {A,B,C,D}, f, g}

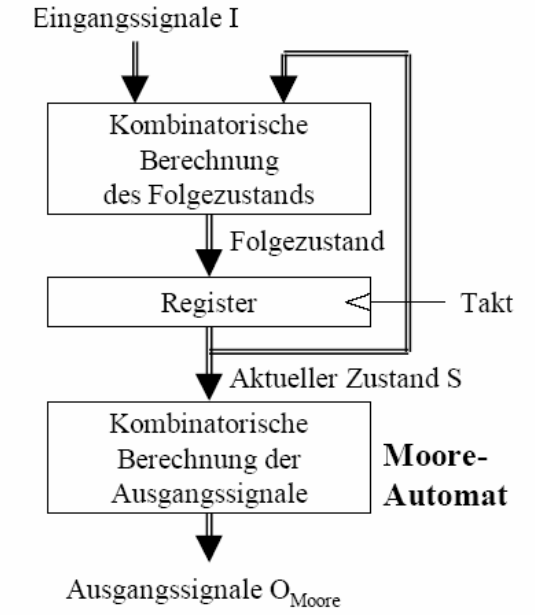




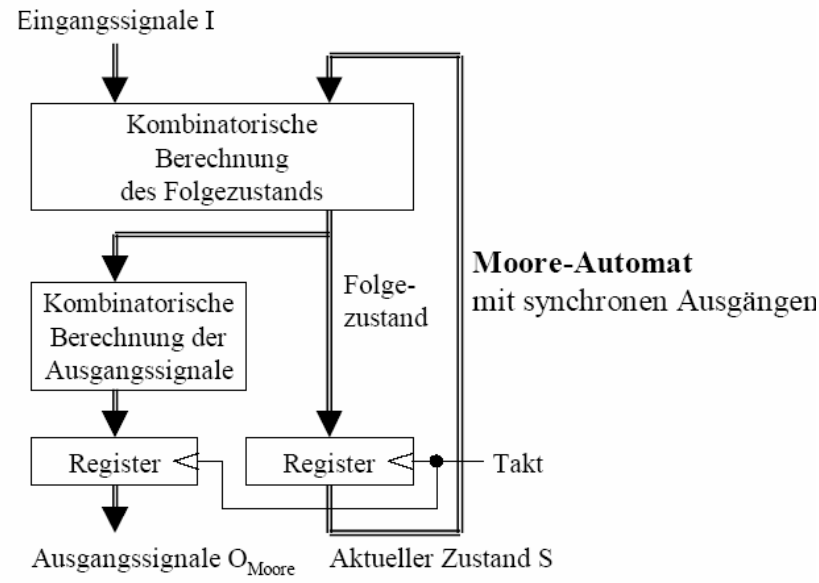
Beispiel Moore-Automat A2



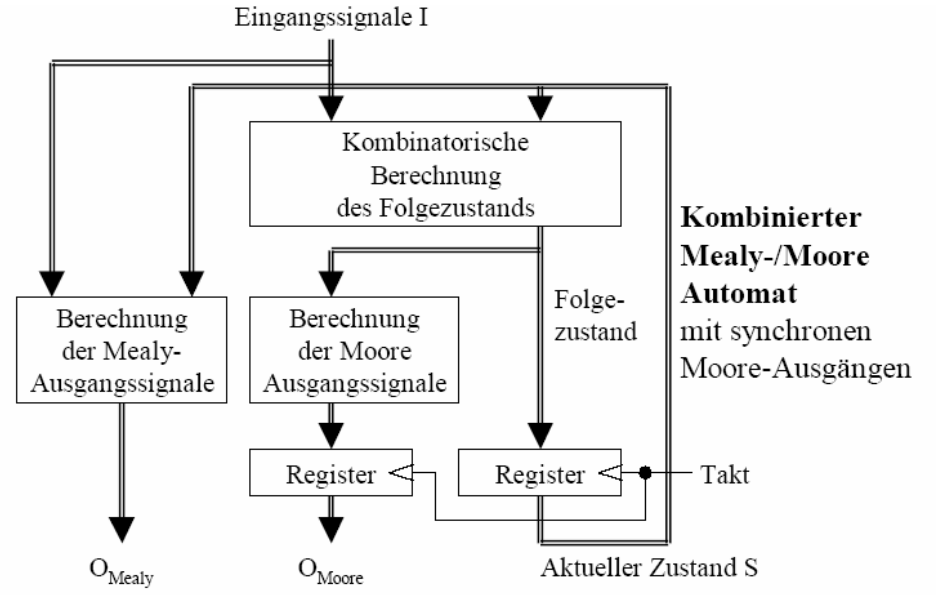
Der Pfeil auf den ersten Zustand bezeichnet den Anfangszustand.



Da beim Moore-Automaten die Ausgangssignale OMoore nur vom aktuellen Zustand abhängen, kann man diese auch in Abhängigkeit des Folgezustands berechnen. Die errechneten Ergebnissignale stehen dann schon vor der aktiven Taktflanke zur Verfügung und müssen über zusätzliche Flip-Flops an die Ausgangssignale geführt
werden. Dies führt logisch zu den gleichen Ausgangssignalen wie beim ersten Vorgehen, diese stehen jedoch sofort nach der aktiven Taktflanke und synchron mit den Zustandssignalen zur Verfügung:



Mögliche Hazards bei der Berechnung der Ausgangssignale treten in dieser Anordnung vor dem Ausgangsregister auf und werden somit nicht auf die Ausgangssignale weitergegeben.



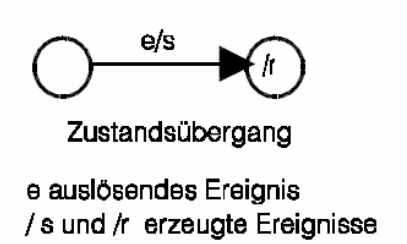
19.1.4 Unvollständige Automaten

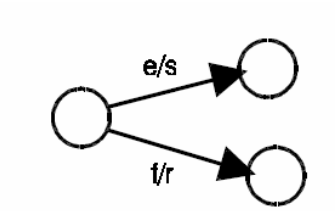
Bisher wurden nur Automaten betrachtet, für die Zustands- und Ausgabefunktion vollständig definiert sind. Zu jedem Wertepaar, bestehend aus Eingabezeichen x und gegenwärtigem Zustand s, ist ein nächster Zustand s‘ und ein Ausgabezeichen y definiert. Oft jedoch sind die Zustands- und/oder die Ausgabefunktion nicht vollständig definiert, entweder weil gewisse Wertepaare (x,s) nicht auftreten können oder weil die Zustandsübergänge für gewisse Wertepaare ohne Bedeutung sind und daher nicht festgelegt wurden. Z.B. ist bei einem Schaltwerk die Anzahl der Zustände immer eine Zweierpotenz, egal ob alle diese Zustände benötigt werden. Die nicht benötigten Zustände sind vom Anfangszustand aus nicht erreichbar.
Deshalb können die Übergänge zwischen diesen Zuständen beliebig sein. Bei unvollständigen

Automaten ist zwischen anwendbaren und nicht anwendbaren Eingabefolgen zu unterscheiden.
In der Praxis ist darauf zu achten, ob durch irgendwelche Fehlerfälle nicht doch Zustände erreicht werden können, für die es keine Folgedefinition gibt (hang up, dead lock)!

19.2 Statecharts

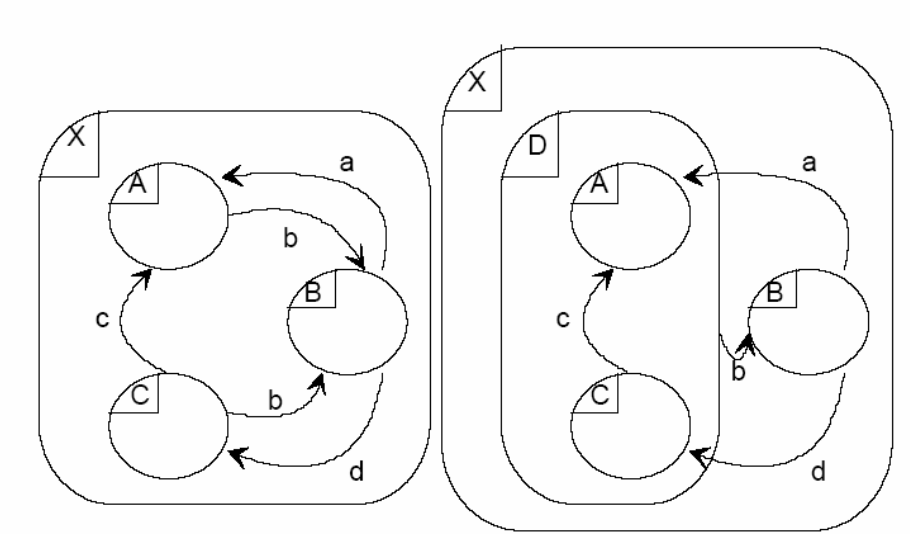
Einfache Statecharts beschreiben endliche Automaten. Damit lassen sich dann komplexere Statecharts gewinnen. Diese Statecharts sind äquivalent zu endlichen Automaten. Sie vereinfachen jedoch die Modellierung erheblich.



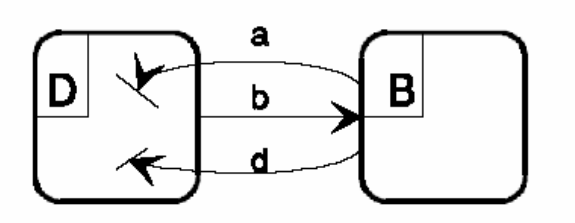


Ein Zustand im Chart ist elementar oder selbst wieder ein Statechart. Statecharts sind also hierarchisch und parallel zerlegbar. Das Zusammenfassen von Zuständen erzeugt Superstates/ODER-Zustand. Der ursprüngliche Automat kann sich in höchstens einem Zustand des Superstates befinden.

19.3 Superstate



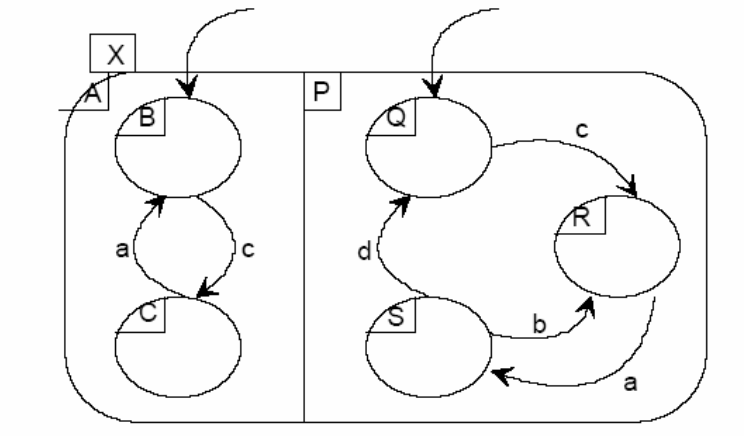
Der Superzustand D faßt in diesem Beispiel die Zustände A und C zusammen:



19.3.1 Nebenläufigkeit

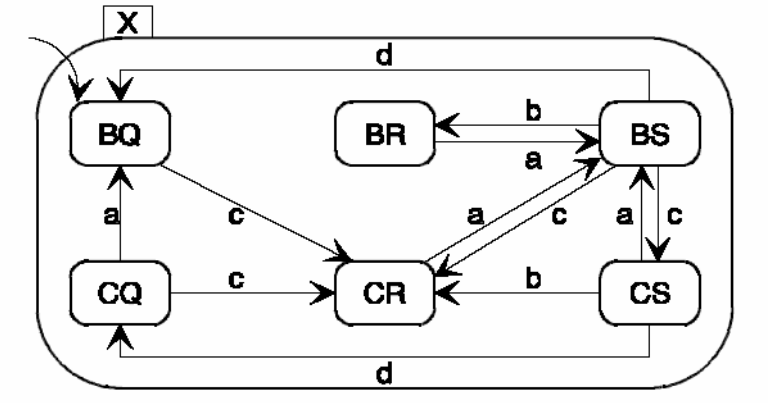
Nebenläufigkeit, d.h. gleichzeitige Zustandsübergänge von Teilsystemen des modellierten Gesamtsystems, lassen sich durch sogenannte UND-Zustände (AND-Charts) darstellen, wobei das System, wenn es sich in einem UND-Zustand befindet, genau einen Zustand in jedem Teilautomaten einnimmt. Eine AND-Chart repräsentiert somit ein Kartesisches Produkt von Zustandsmengen. Ein Superstate kann nun z.B. durch einen solchen UND-Zustand gegeben sein, der sich seinerseits wieder in ODER-Zustände zerlegen läßt. Die Struktur eines Superstate läßt sich somit durch einen Booleschen Ausdruck beschreiben.

Der Superstate X besteht aus den Superzuständen A und P, die nebenläufig eingenommen werden: X = A*P = (B+C)*(Q+S+R). Wenn das System in den Zustand X übergeht, geht es in den Zustand (B,Q), d.h. das erste Teilsystem nimmt Zustand B, das zweite Zustand Q ein. Die Übergänge der Teilsysteme (d.h. innerhalb der Superstates A und P) können unabhängig oder voneinander abhängig erfolgen.



19.3.2 UND-Zustand

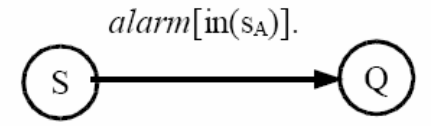
Beispielsweise bewirkt das externe Ereignis c, daß - wenn sich das System im Anfangszustand (B,Q) befindet - es in den Zustand (C,R) wechselt (nebenläufige Übergänge der Teilautomaten). Auf diese Weise können die Teilautomaten synchronisiert werden. Wenn sich das System jedoch im Zustand (B,S) befindet, bewirkt das Ereignis c, daß der erste Teilautomat nach C wechselt und der zweite Teilautomat seinen Zustand beibehält, also einen Übergang nach (C,S).



Äquivalenter (unvollständiger) Automat

19.3.3 Guard

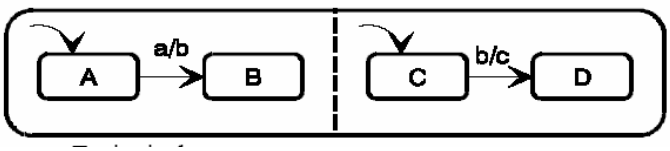
Es kann sein, daß der Zustandsübergang eines Teilautomaten bestimmte Übergänge in anderen Teilautomaten unterbindet oder bestimmte Ereignisse nur in bestimmten Zuständen wirken können. Diese Abhängigkeiten können durch sogenannte Guards (Wächter) ausgedrückt werden. Ein Guard ist ein logischer Ausdruck, der einem Zustandsübergang zugeordnet ist und erfüllt sein muß, damit dieser Zustandsübergang stattfinden kann. Einfache Guards haben die Form:
Ereignis[in(Zustand)], oder Ereignis[NOT in(Zustand)]
z.B.: Zustandsübergang des Teilautomaten B:



Das Ereignis 'alarm' bewirkt nur dann den Zustandsübergang von S nach Q in B, wenn Teilautomat A im Zustand sA ist. In anderen Worten, bevor Teilautomat B auf das Alarmsignal reagiert, liest (prüft) er das Zustandsregister von Teilautomat A.

19.3.4 Rundspruch/broadcast

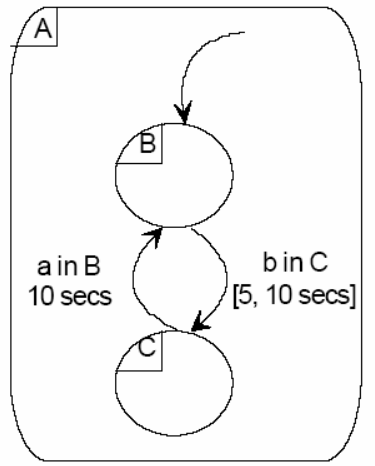
Ereignisse lösen Zustandsübergänge aus. Ereignisse können aber auch bewirken, daß andere Ereignisse erzeugt werden - äußere und innere Ereignisse. Solche äußeren Ereignisse sind die Ausgaben des Systems, innere Ereignisse können wiederum Zustandsübergänge auslösen. Innere Ereignisse sind "überall im System bekannt". (Sie werden durch Rundspruch bekannt gemacht). Auf diese Weise kann ein Ereignis z.B. eine Kettenreaktion von Zustandsübergängen auslösen. Ereignis a generiert Ereignis b und dieses wiederum



19.3.5 History

19.3.6 Zeitangaben

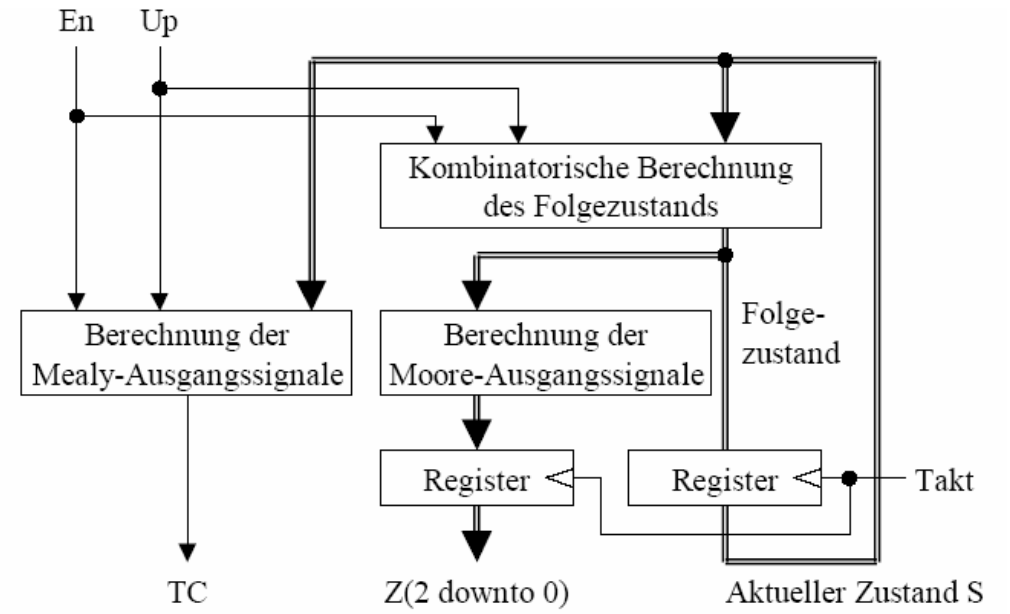
Da Statecharts vor allem auch dafür verwendet werden, reaktive Systeme zu modellieren, muß es möglich sein, Zeitverhältnisse anzugeben. Es wird dazu eine globale Uhr orausgesetzt, die von jedem Zustand aus gelesen werden kann. Logische Ausdrücke können angegeben werden, die besagen, wann ein Zustandsübergang erfolgt. So findet Übergang a unten immer dann statt, wenn der Automat genau 10s im Zustand B war, während Übergang b stattfindet, wenn der Automat wenigstens 5 Sekunden, aber nicht mehr als 10 Sekunden, im Zustand C ist.



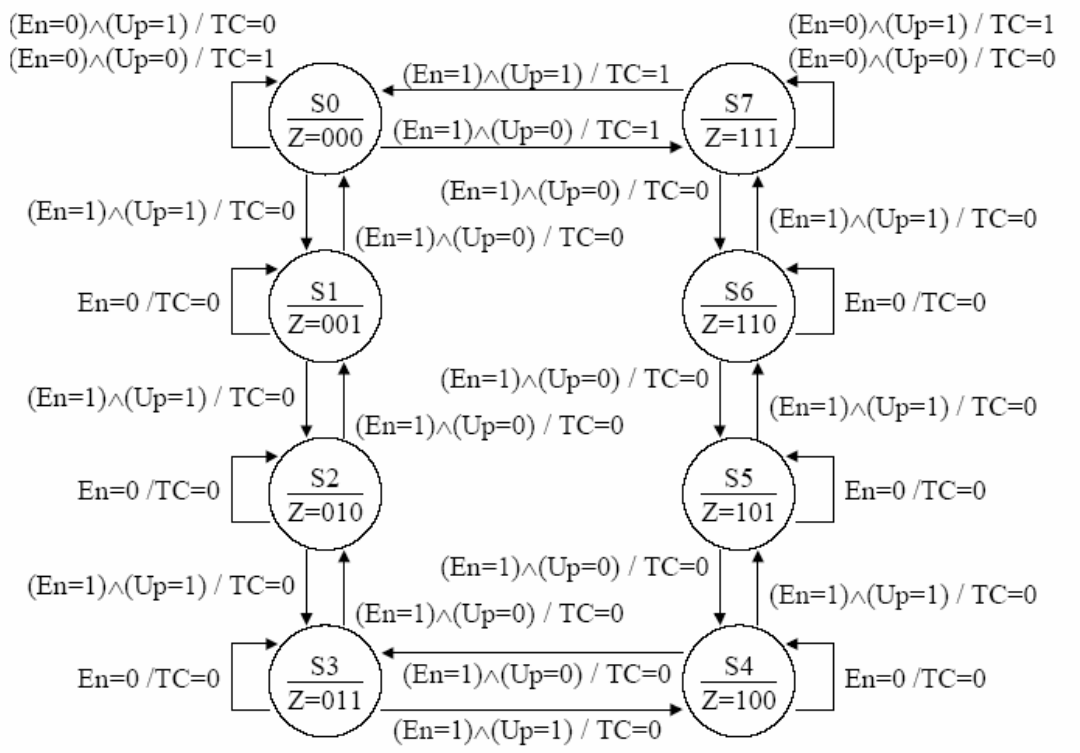
Beispiele zum Entwurf endlicher Automaten
Entwurf eines 3-Bit Auf-/Abwärtszählers mit Freigabeeingang
Ein 3-Bit Auf-/Abwärtszähler soll als endlicher Automat entworfen und realisiert werden. Dies wird von der Schaltungstechnik her gesehen einen deutlich höheren Aufwand ergeben, als beim vorgestellten, optimierten Zählerentwurf. Der Entwurf konzentriert sich jedoch nun nicht mehr auf die Schaltungstechnik, sondern auf die Funktionalität. Damit ist der Entwurf flexibler und kann bei Änderungswünschen gut angepasst werden.
Der Zähler soll zwei Eingänge und zwei Ausgänge erhalten:

Signal	**Typ**	**Beschreibung**
En	Eingang	Freigabeeingang: Wenn das Signal auf '1' liegt, zählt der Zähler, bei '0' wird der Zählvorgang angehalten.
Up	Eingang	Zählrichtung: Liegt das Signal auf '1', wird aufwärts gezählt, ansonsten abwärts.
TC	Ausgang	Zählende: Erreicht der Zähler beim Hochzählen den größten Wert 7 oder beim Abwärtszählen den Wert 0, wird der Ausgang aktiviert, d.h. auf '1' gesetzt.
Z(2 downto 0)	Ausgang	Zählerstand: Über diesen Ausgangsvektor wird der Zählerstand ausgegeben.

Der Zähler soll als kombinierter Mealy/Moore-Automat entworfen werden, damit ergibt sich die folgende Schaltung:



Das Zustandsdiagramm modelliert den Zähler mit 8 Zuständen S0 bis S7. Jedem Zustand wird ein zugehöriger Zählerstand Z zugeordnet. Damit ist der Ausgang Z ein Moore-Ausgang.



Der TC-Ausgang soll sich sofort in Abhängigkeit des Up-Eingangs ändern. Steht beispielsweise der Zähler im Zustand S0 und der Eingang Up ändert sich von '1' (hochzählen) auf '0' (abwärtszählen), dann soll sofort der TCAusgang das Zählende mit TC= '1' markieren. Im Zustandsdiagramm ist TC daher als Mealy-Ausgang eingetragen. Mit dem En-Eingang wird der Zähler freigegeben. Solange En= '0' gesetzt ist verharrt der Zähler in einem Zustand, bei En= '1' zählt der Zähler und die Zählrichtung hängt vom Up-Eingang ab.
Aus dem Zustandsdiagramm lässt sich die Funktionstabelle für den Automaten erstellen. Diese umfaßt 32 mögliche Kombinationen aus Zustands- und Eingangswerten und ordnet jeder Kombination einen Folgezustand und zugehörige Werte der Mealy-Ausgänge zu:

Zustand	En	Up	Folgezustand	TC
S0	0	0	S0	1
S0	0	1	S0	0
S0	1	0	S7	1
S0	1	1	S1	0
S1	0	-	S1	0
S1	1	0	S0	0
S1	1	1	S2	0
S2	0	-	S2	0
S2	1	0	S1	0
S2	1	1	S3	0
S3	0	-	S3	0
S3	1	0	S2	0
S3	1	1	S4	0
S4	0	-	S4	0
S4	1	0	S3	0
S4	1	1	S5	0
S5	0	-	S5	0
S5	1	0	S4	0
S5	1	1	S6	0
S6	0	-	S6	0
S6	1	0	S5	0
S6	1	1	S7	0
S7	0	0	S7	0
S7	0	1	S7	1
S7	1	0	S6	0
S7	1	1	S0	1

Die Moore-Ausgänge ergeben sich direkt aus den Zuständen, somit werden diese durch eine zweite Funktionstabelle beschrieben:

Zustand	Z(2 downto 0)
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
S6	110
S7	111

Will man den Automaten in Hardware realisieren, muß nun den Zustandswerten eine Kodierung zugeordnet werden. Häufig werden bei Zählern die Ausgangswerte Z auch als Zustandskodierung verwendet, dies ist jedoch nicht zwingend. Eine 1 of N Kodierung, bei der immer nur ein Bit zu '1' gesetzt ist, führt beispielsweise in vielen Fällen zu einer einfachen Kombinatorik.
Nach Wahl einer Zustandskodierung (z.B. Binär, Gray-Code oder 1 of N) können aus den beiden Tabellen die kombinatorischen Funktionen ermittelt werden, welche aus dem aktuellen Zustand und den Eingangswerten den Folgezustand, den Mealy-Ausgang TC und die Moore-Ausgänge Z(2 downto 0) berechnen.
Die Ermittlung der kombinatorischen Gleichungen bleibt dem Leser als Übung überlassen.