Einführung in die Technische Informatik

Zusammenfassung Buch zweite Auflage

1.Hardware

Die Struktur eines Computers setzt sich aus folgenden Komponenten zusammen:

Rechenwerk: Transfer und Verarbeitung der Daten

Speicherwerk: Speicherung der Daten und Programme

Steuerwerk: Koordination der einzelnen Komponenten

Ein-/Ausgabeeinheiten: Informationsaustausch mit Peripheriegeräten

Bussystem: Datenaustausch zwischen den einzelnen Teilen

Programmierung

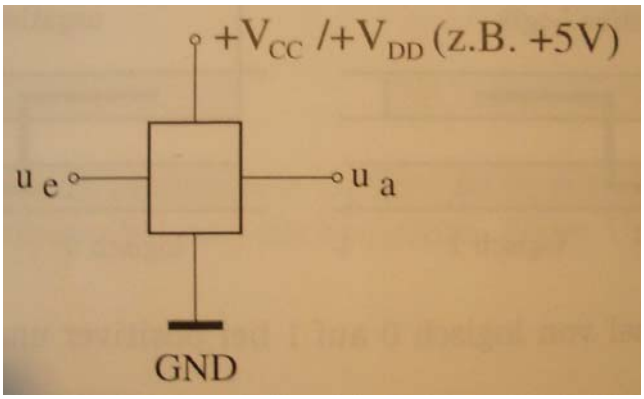
Übersetzung: Ein Compiler Programm übersetzt den Source Code eines Programms in Maschinen-Instruktionen die vom Computer exekutiert werden können.

Interpretation: Ein Programm wird zur Laufzeit Zeile für Zeile eingelesen und die entsprechende Folge von Maschinenbefehlen wird online exekutiert.

2.Logische Schaltungen

2.1 Grundbegriffe

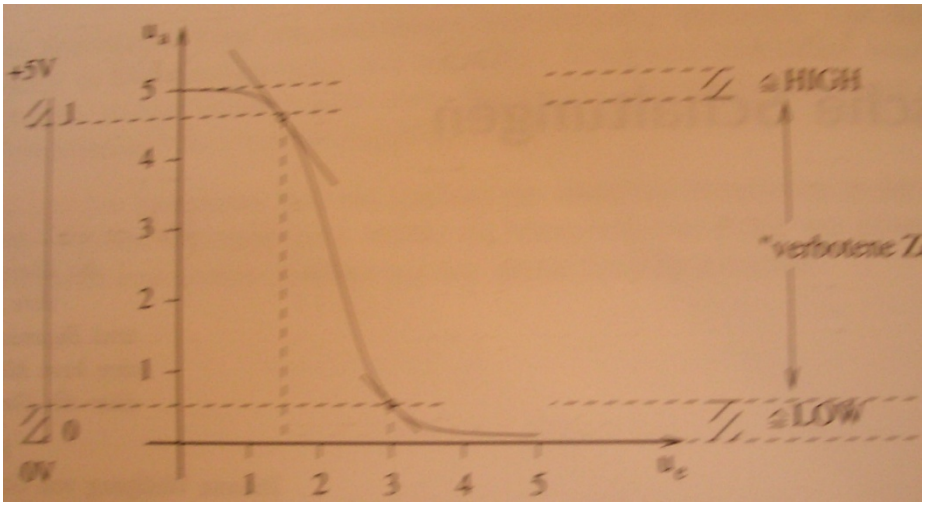
Die Versorgungspannung einer elektronischen Schaltung beträgt üblicherweise +5V oder +12V. Eingangsspannung u_e und Ausgangsspannung u_a liegen stets zwischen 0V und der Versorgungsspannung.



Positive/Negative Logik

Positive Logik: $U_{High} = 1$ und $U_{Low} = 0$

Negative Logik: $U_{High} = 0$ und $U_{Low} = 1$



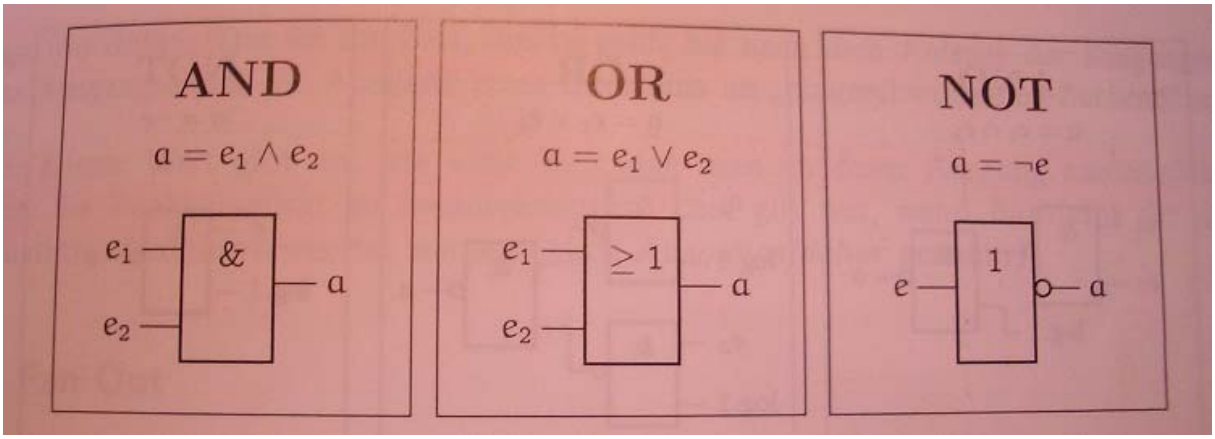
Die Spannungswerte im Bereich der „Verbotenen Zone“ sind nicht zulässig.

Operatoren

In elektronischen Schaltungen werden Bauteile verwendet die die Mathematischen Operatoren

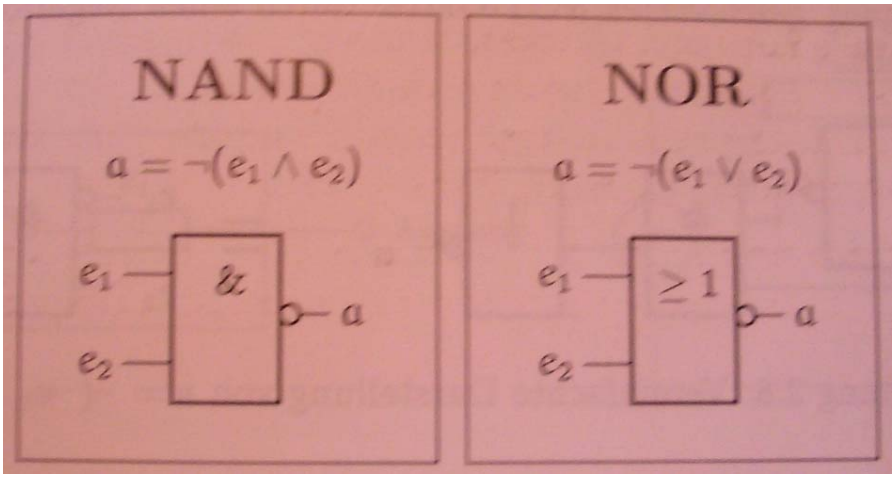
UND ^
ODER v
NICHT ¬

verwendet.

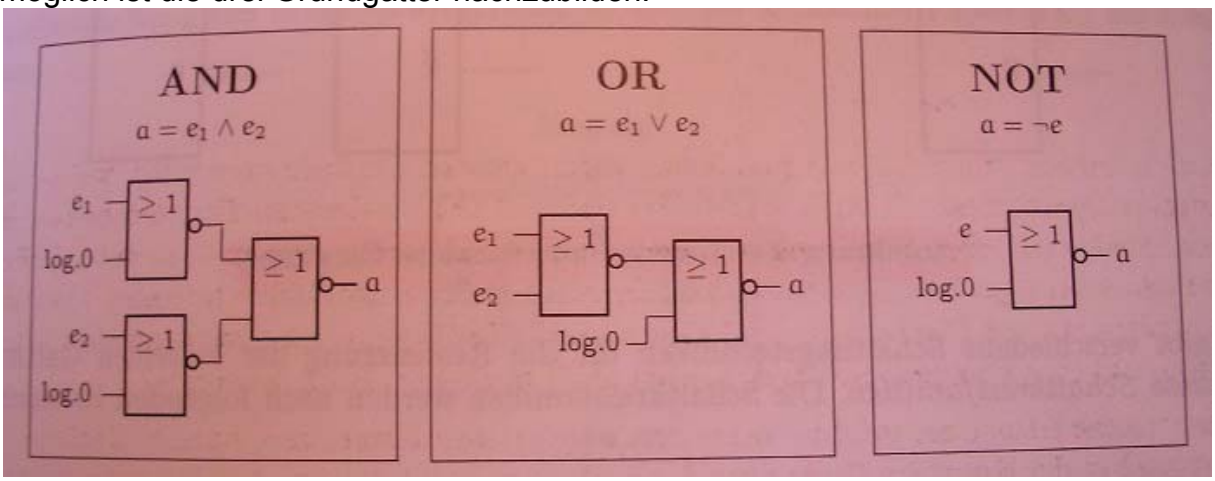


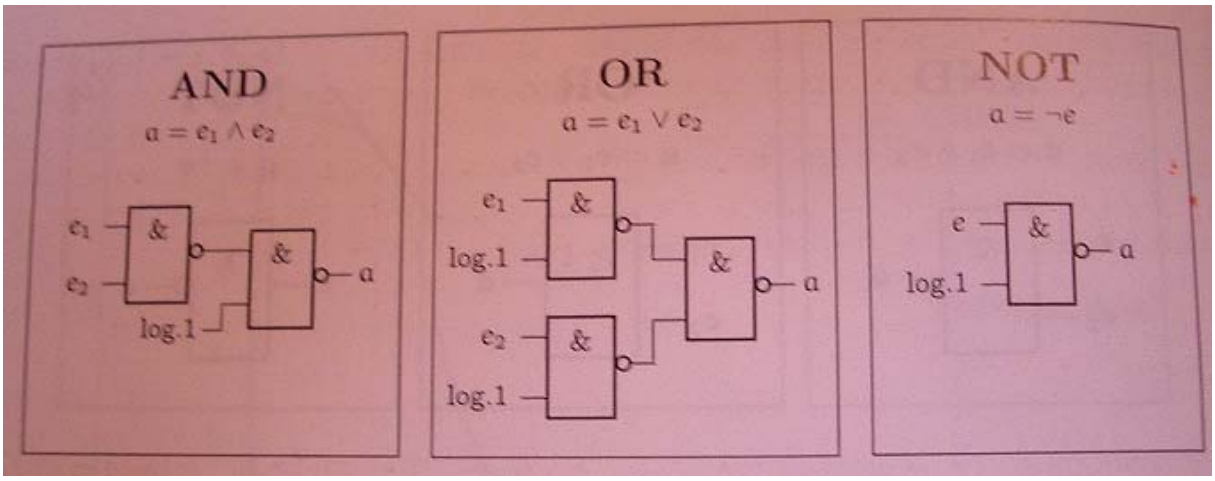
Die drei Grundgatter UND, ODER, NICHT.

Weiters finder folgende Gatter häufig verwendung:



NAND und NOR nennt man auch universelle Gatter, da es mit jedem dieser Gatter möglich ist die drei Grundgatter nachzubilden:





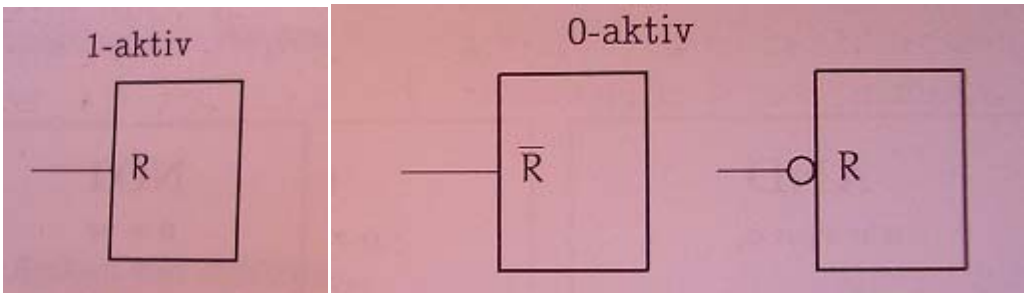
Mit Hilfe der Boolschen Algebra und den de Morganschen Gesetzen, kann man die Korrektheit von Gatterschaltungen nachvollziehen.

De Morgansche Gesetze:

¬(X v Y) = ¬X ^ ¬Y

¬(X ^ Y) = ¬X v ¬Y

Man unterscheidet bei Gattern zwischen 1-aktiven und 0-aktiven Eingängen, 1-akitve Eingänge werden beim anliegen von 1 aktiv und 0-aktive Eingänge werden beim anliegen von 0 aktiv.



Power dissipation: Verlustleistung eines Gatters

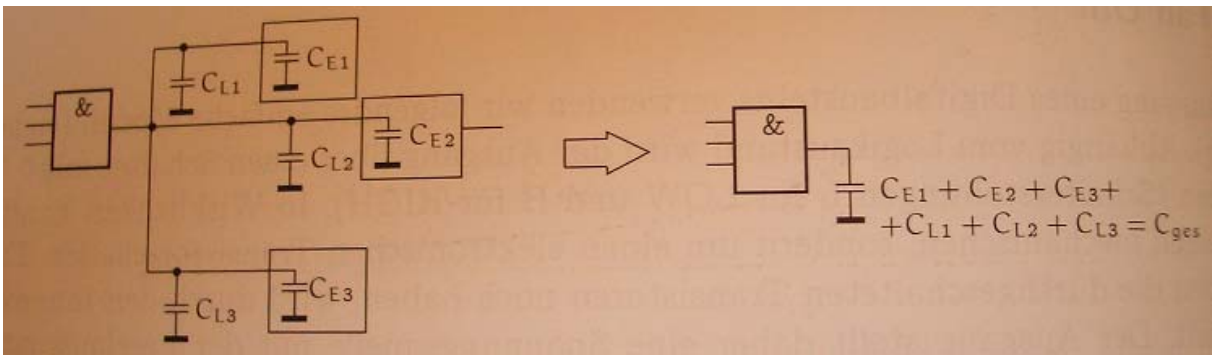
Propagation delay: Zeit die vergeht bis das Eingangssignal am Ausgang im „eingeschungenen Zustand“ anliegt

<u>Fan-out/Fan-in</u>

Fan-out: Bezeichnet die Anzahl wie viele Eingänge an einen Ausgang angehängt werden dürfen ohne das die Funktionsweise des Gatters beeinträchtigt wird.

Fan-in: Bezeichnet die Anzahl der Ausgänge die an einen Eingang zusammenlaufen dürfen ohne das die Funktionsweise des Gatters beeinträchtigt wird.

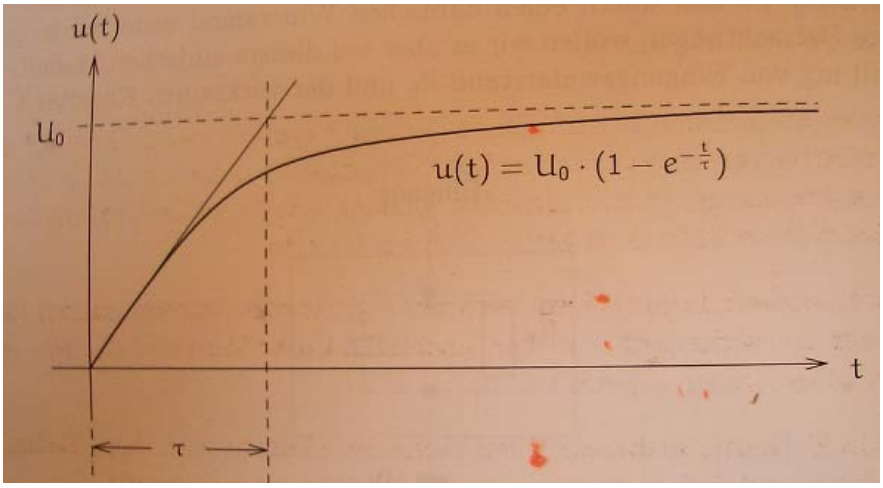
Werden mehrere Ausgänge an einen Eingang geschaltet, wird dieser Ausgang mit der Gesamtkapazität der Eingänge belastet:



Bei jedem Umschaltvorgang muss die Kapazität über den Innenwiederstand des Ausgangs auf- bzw. entladen werden.

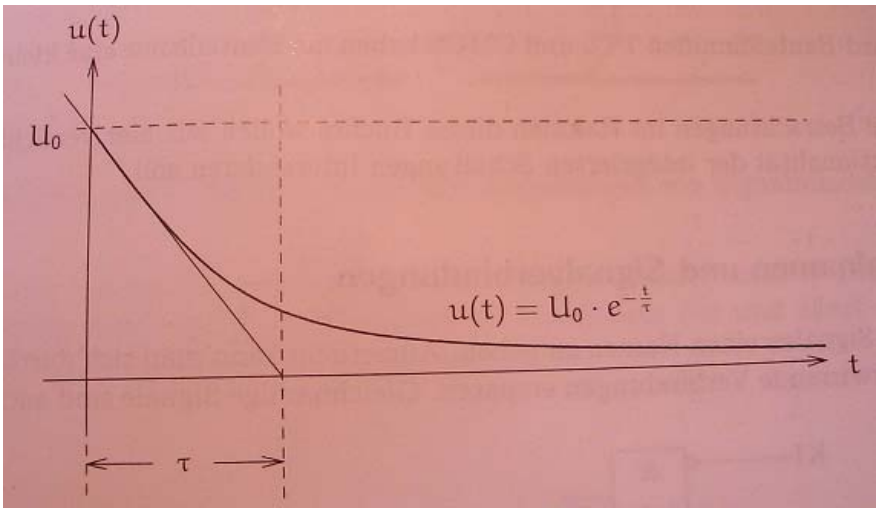
Aufladevorgang:

$u_c(t) = U_0 * (1 – e^{-t/T})$



Entladevorgang:

$u_c(t) = U_0 * e^{-t/T}$



T setzt sich zusammen aus R (Innenwiederstand) mal C_{ges} (Gesamtkapazität).

$T = R * C_{ges}$

<u>Schaltkreisfamilien</u>

TTL (Transistor-Transistor-Logic): großes Fan-out, große Gatter Laufzeit, wird kaum noch verwendet.
ECL (Emitter Coupled Logic): geringste Gatterlaufzeiten, hohe Störsicherheit, hohe Kosten, große Verlustleistungen
MOS (Metal-Oxid Semiconductor): meist nur bei hoch integrierten Schaltungen
CMOS (Complementary MOS): geringste Leistungsaufnahme, Standard im PC-Bereich

2.2 Realisierung von Funktionen

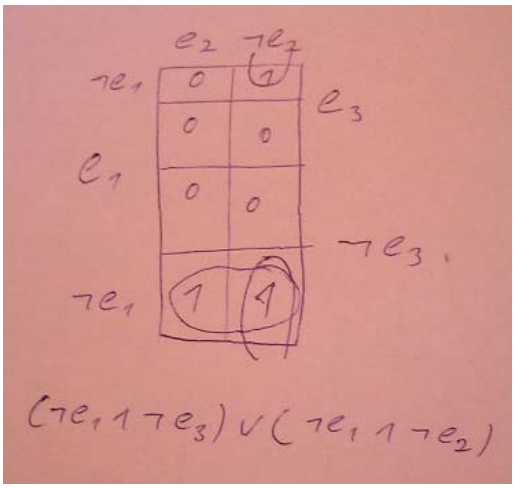
Es soll eine Schaltung mit drei Eingängen (e1, e2, e3) realisiert werden, am Ausgang wird 1 ausgegeben wenn die Zahl an den Eingängen kleiner als 3 ist.

dez	binär	e_1	e_2	e_3	a
0	000	0	0	0	1
1	001	0	0	1	1
2	010	0	1	0	1
3	011	0	1	1	0
4	100	1	0	0	0
5	101	1	0	1	0
6	110	1	1	0	0
7	111	1	1	1	0

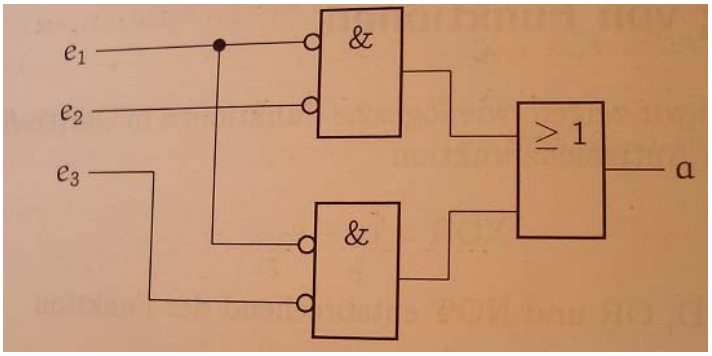
Aus der Wahrheitstabelle erhält man die disjunktive Normalform:

a = f(e1, e2, e3) = (¬e1 ^ ¬e2 ^ ¬e3) v (¬e1 ^ ¬e2 ^ e3) v (¬e1 ^ e2 ^ ¬e3)

Vereinfachung im KV-Diagramm:



a = (¬e1 ^ ¬e3) v (¬e1 ^ ¬e2)

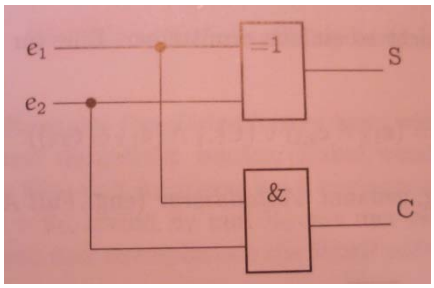


Realisierung der Funktion als Gatterschaltung

Halbaddierer

Addierer sind logische Schaltungen die zwei Binärzahlen miteinander addieren. Ein Halbaddierer ist die einfachste Form, er kann zwei einstellige Binärzahlen addieren, ein Übertrag wird jedoch nicht berücksichtigt.

e_1	e_2	$e_1 + e_2$
0	0	00
0	1	01
1	0	01
1	1	10



Die Boolschen Funktionen dieser Schaltung schauen folgendermaßen aus:

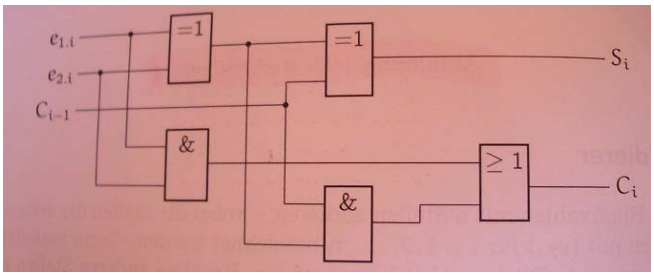
S = e1 # e2 (# = Antivalenzfunktion)

C = e1 ^ e2

Volladdierer

Ein Volladdierer ist in der Lage Binärzahlen mit n stellen zu addieren, im gegensatz zum Halbaddierer wird beim Volladdierer auch der Übertrag der nächstniedrigeren Stelle berücksichtigt.

$e_{1.i}$	$e_{2.i}$	C_{i-1}	C_i	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Die disjunktiven Normalformen dieser Funktionen sehen damit folgendermaßen aus:

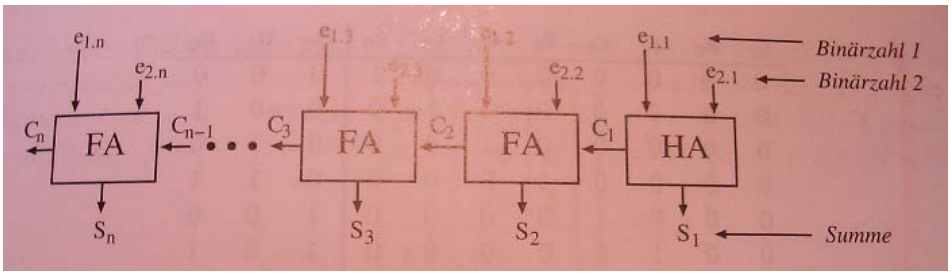
S_i = (¬e1.i ^ ¬ e2.i ^ Ci-1) v (¬e1.i ^ e2.i ^ ¬Ci-1) v (e1.i ^ ¬e2.i ^ ¬ Ci-1) v (e1.i ^ e2.i ^ Ci-1)

Ci = (¬e1.i ^ e2.i ^ Ci-1) v (e1.i ^ ¬e2.i ^ Ci-1) v (e1.i ^ e2.i ^ ¬Ci-1) v (e1.i ^ e2.i ^ Ci-1)

Durch Vereinfachung erhält man folgende Form:

Si = e1.i # e2.i # Ci-1
Ci = (e1.i ^ e2.i) v (Ci-1 ^ (e1.i # e2.i))

Durch hintereinander Schalten mehrerer Addierer, lassen sich sogenannte Paralleladdierer realisieren, sie sind in der Lage beliebig lange Binärzahlen zu addieren bzw. mit dem Einer- und Zweierkomplement zu subtrahieren.



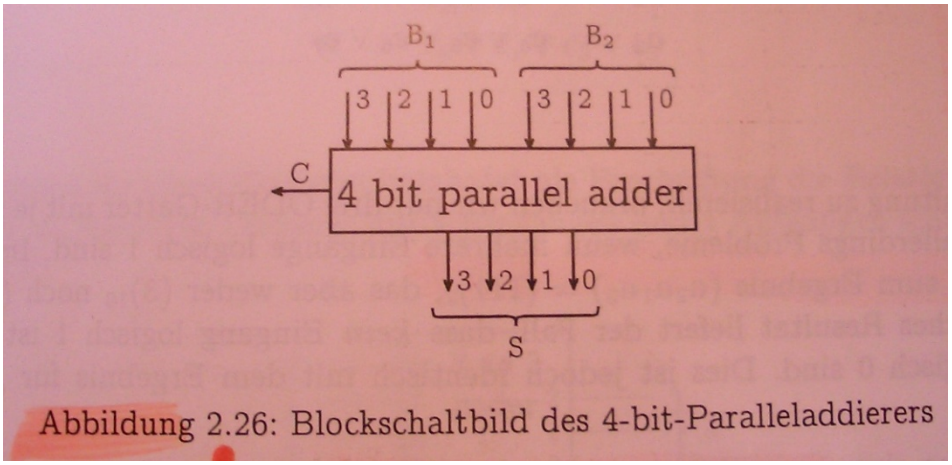


Abbildung 2.26: Blockschaltbild des 4-bit-Paralleladdierers

Codierer

Ein Codierer ist ein Schaltung die bei n Eingängen e_i (i = 0, 1, … , n -1) genau m = [ld(n)] Ausgänge a_i (i = 0, 1, … , m -1) besitzt.
Unter der Bedingung das immer nur ein Eingang logisch 1 sein kann, wandelt der Codierer die Bitfolge die and den Eingängen anliegt in eine Binärzahl an den Ausgängen um.

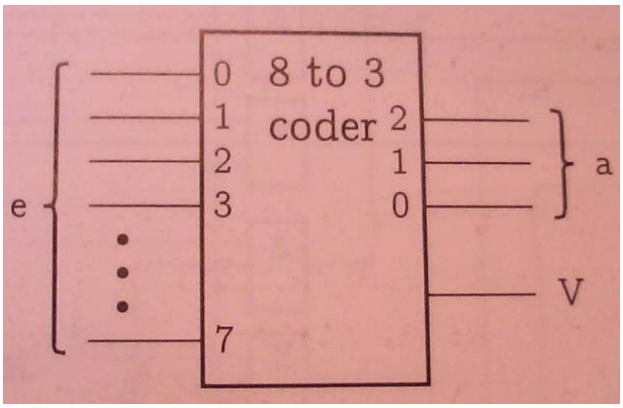
e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0	a_2	a_1	a_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Dabei ergeben sich jedoch Probleme wenn mehrere Eingänge logisch 1 sind, sind z.B. e3 = e6 = 1 führt das zum Ergebnis (a2a1a0) = $(111)_2$ das entspricht jedoch weder $(3)_{10}$ noch $(6)_{10}$
Durch die Verwendung eines prioritätsgesteuerten Codierers werden diese Probleme vermieden. Bei dieser Schaltung wird nur jener Eingang beachtet dessen Index am größten ist. Für die X kann im KV Diagramm beliebig 0 oder 1 eingesetzt werden, V zeigt an ob mindestens ein Eingang logisch 1 also gültig ist.

e_3	e_2	e_1	e_0	a_1	a_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

a0 = (e1 ^ ¬e2) v e3
a1 = e2 v e3
V = e0 v e1 v e2 v e3

Blockschaltbild eines 8 zu 3 Codierers:



Decodierer

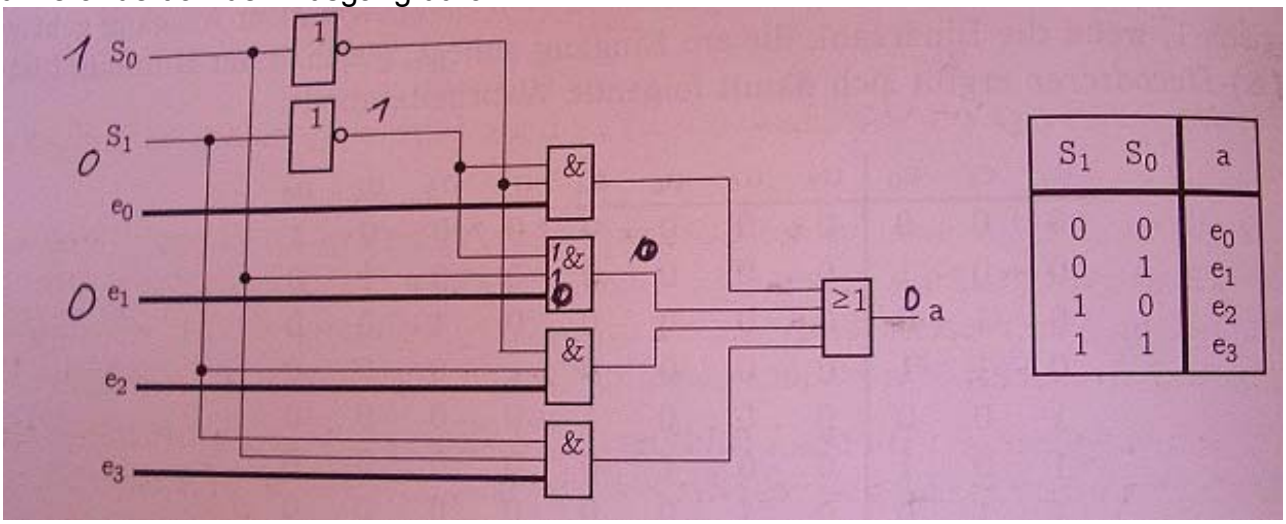
Das Decodierer ist das Gegenstück zum Codierer, er ist eine Schaltung mit n Ausgängen und m = [ld(n)] Eingängen. Ein Ausgang geht genau dann auf logisch 1, wenn die Binärzahl, die am Eingang anliegt, gleich seiner Nummer i ist.

e_2	e_1	e_0	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

E	e_1	e_0	a_3	a_2	a_1	a_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

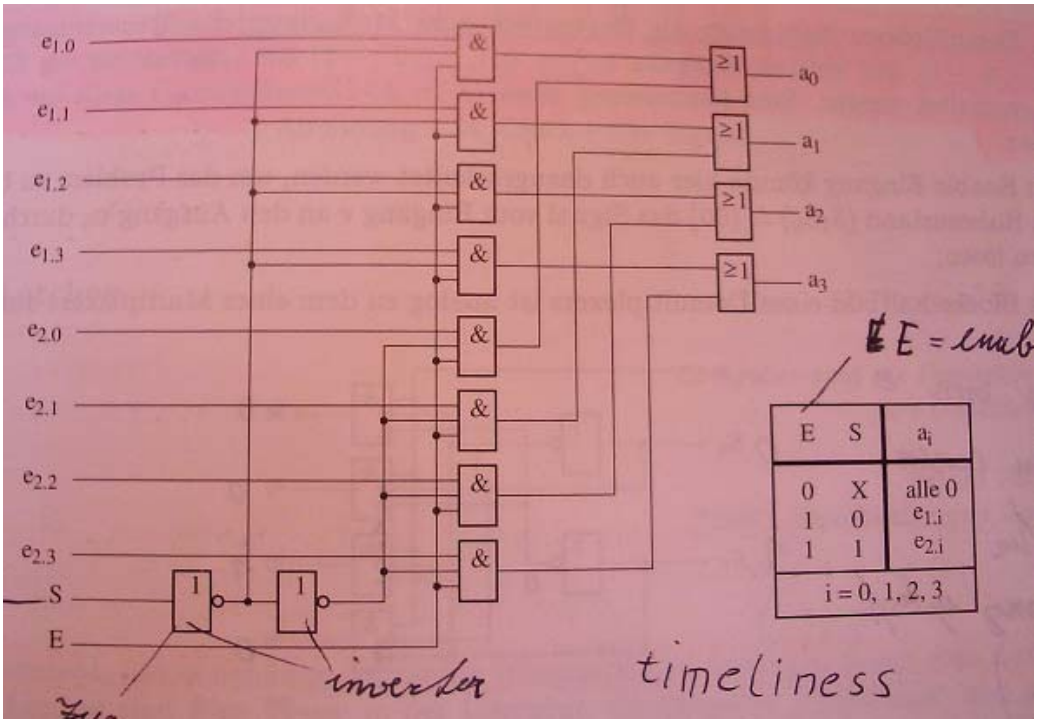
Multiplexer

Ein Multiplexer ist einer Schaltung die aus m binären Eingängen e_i (i = 0, 1, ... , m-1) jenen auswählt, dessen Nummer mit der Zahl (Sn-1 … S1 S0) übereinstimmt, die an den Steuervariablen S_j (j = 0,1, … , n-1) anliegt und schaltet dessen Information unverändert an den Ausgang durch.

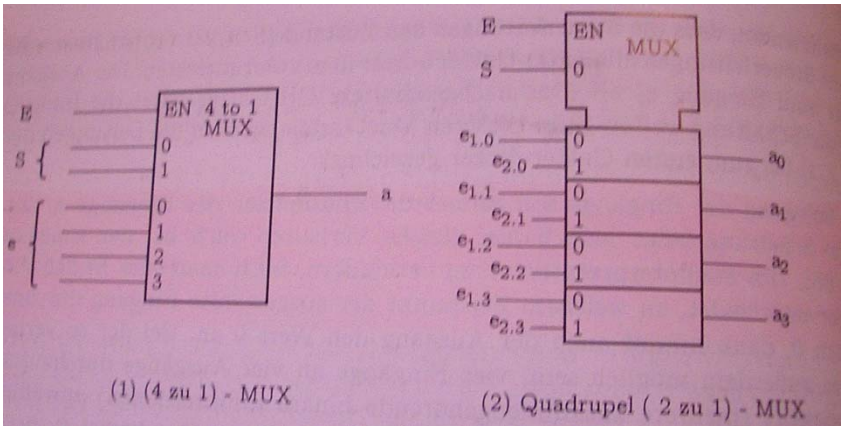


S_1	S_0	a
0	0	e_0
0	1	e_1
1	0	e_2
1	1	e_3

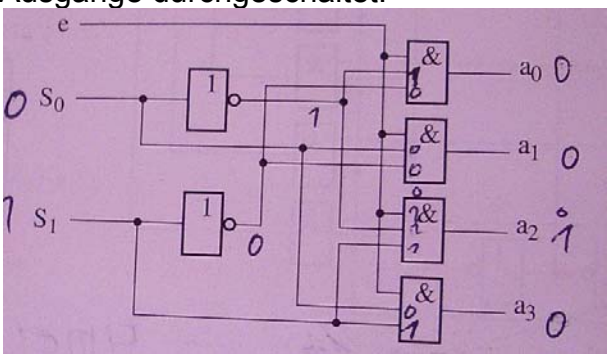
Ohne Belegung soll am Ausgang immer die Information des Eingangs e_0 anliegen, dies entspricht aber auch der Eingangskombinatiuon (S1S0) = (00), dieses Problem kann durch eine zusätzliche Enable Leitung (E) gelöst werden.



Blockschaltbild eines 4 zu 1 und eines Quadrupel 2 zu 1 Multiplexers:



<u>Demultiplexer</u>
Gegestück zum Multiplexer, der Eingang e wird durch die Steuereingänge S_i auf die Ausgänge durchgeschaltet.



2.3 Sequenzielle Logik

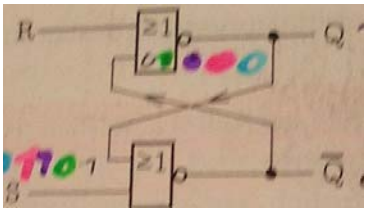
Bei den bisher entworfenen Schaltungen hing das Ergebnis der Ausgänge direkt von den Zuständen der Eingänge ab. Praktisch ist es jedoch von Vorteil, Informationen eine bestimmte Zeit speichern zu können, da nicht immer alle nötigen Daten zur selben Zeit gleich lang zur Verfügung stehen. Dafür benötigt man Speicherelemente. Schaltungen die Speicherelemente enthalten, heissen sequenzielle Schaltungen. Man unterscheidet zwischen synchronen und asynchronen Schaltungen. Bei Synchrone Schaltungen wird ein Zustandswechsel nur zu bestimmten Zeitpunkten durchgeführt, bei asynchronen Schaltungen wird ein Zustandswechsel zu jedem beliebigen Zeitpunkt durchgeführt.

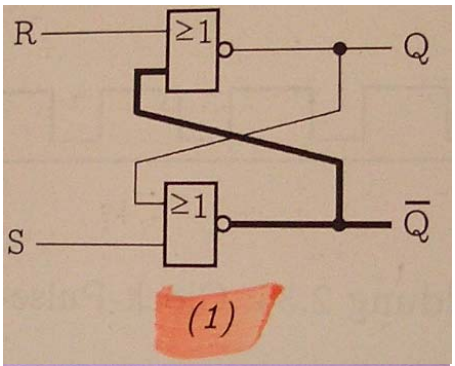
Latches

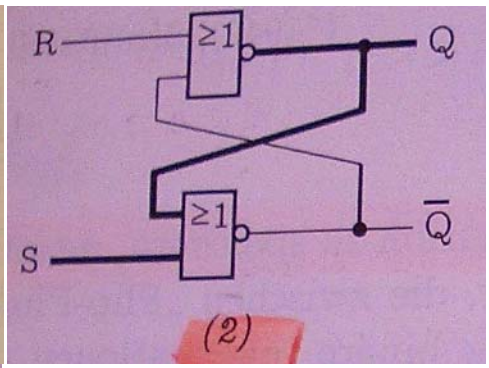
Ein Schaltelement das in der Lage ist binäre Informationen zu speichern nennt man Latch.
Sie können binäre Information, so lange Stromversorgung vorhanden ist, auf unbestimmte Zeit speichern, abfragen und verändern.

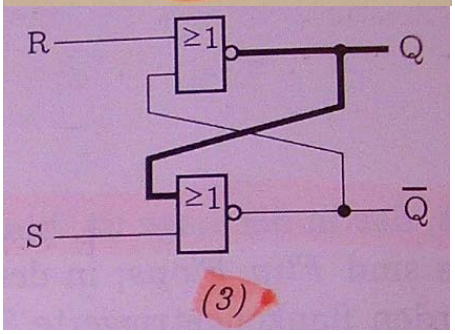
RS-Latch

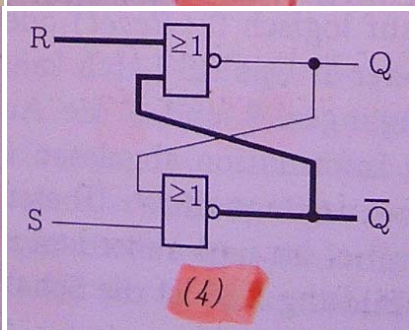
R = Reset Eingang (Löschen)
S = Set Eingang (Setzen)
Q = Ausgang
Q^- = negierter Ausgang





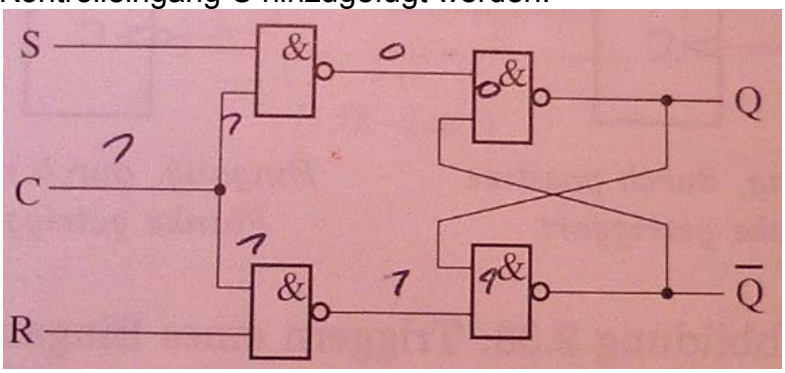






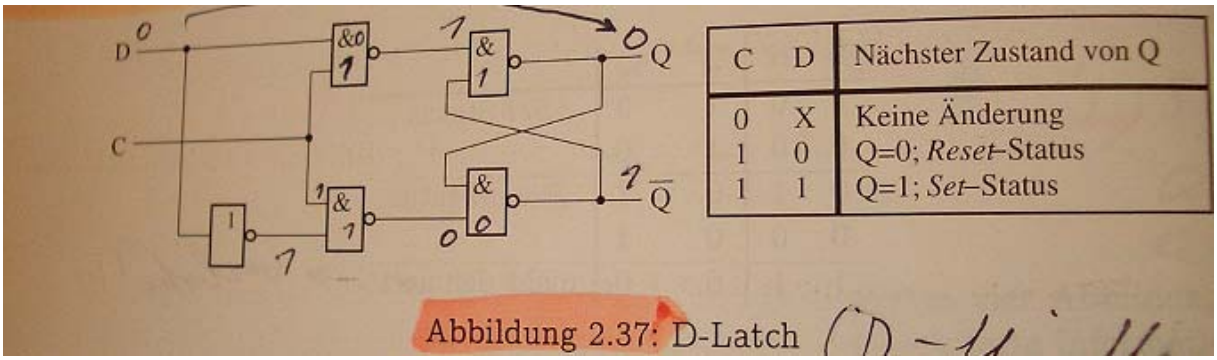
S	R	Q	¬Q	
1	0	1	0	*Set*-Status
0	0	1	0	
0	1	0	1	*Reset*-Status
0	0	0	1	
1	1	0	0	nicht definiert

Um aus einem asynchronen Latch einen synchronen zu machen, kann zusätzlich ein Kontrolleingang C hinzugefügt werden:



C	S	R	Nächster Zustand von Q
0	X	X	Keine Änderungen
1	0	0	Keine Änderungen
1	0	1	Q=0; *Reset*-Status
1	1	0	Q=1; *Set*-Status
1	1	1	Nicht definiert

Der Fall R = S = 1 ist beim RS Latch nicht definiert, durch die Erweiterung auf einen D Latch wird dieser Fall behandelt.



C	D	Nächster Zustand von Q
0	X	Keine Änderung
1	0	Q=0; *Reset*-Status
1	1	Q=1; *Set*-Status

Abbildung 2.37: D-Latch

Weiter ist noch das JK Latch zu erwähnen, hier wird die Situation R = S = 1 anders als beim D-Latch behandelt.

Folgende Überlegungen führen zum Aufbau des JK-Latch:

- Es ist nicht erfoderlich, ein Speicherelement, das bereits den Wert 1 enthält, nochmals zu setzen
- Ein zurückgesetztes Latch muss nicht erneute gelöscht werden

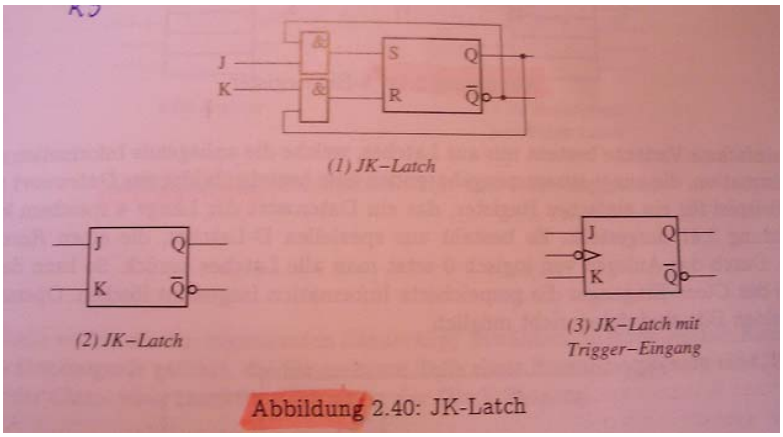


Abbildung 2.40: JK-Latch

J	K	Q_{t-1}	Q_t
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	1	0

Register

Üblicherweise bestehen Informationen aus n zusammengehörenden Bits und sollten auch gemeinsam gespeichert werden.
Die einfachste Variante besteht nur aus Latches, welche die anliegende Information speichern.

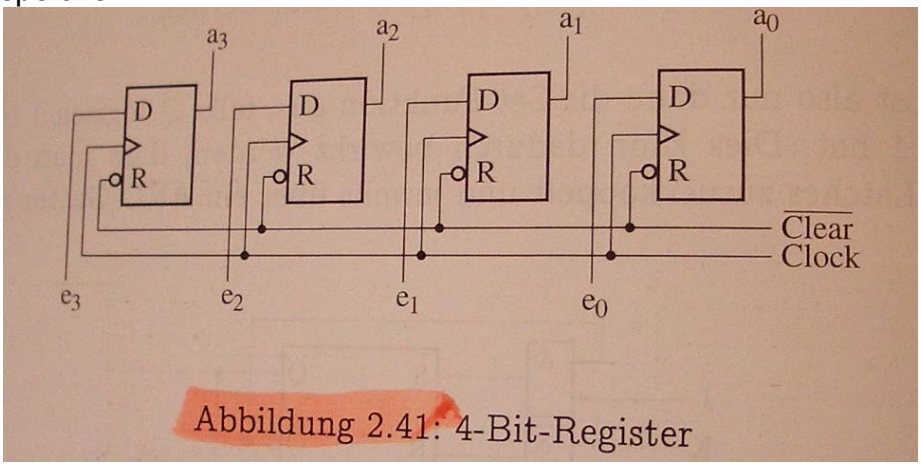
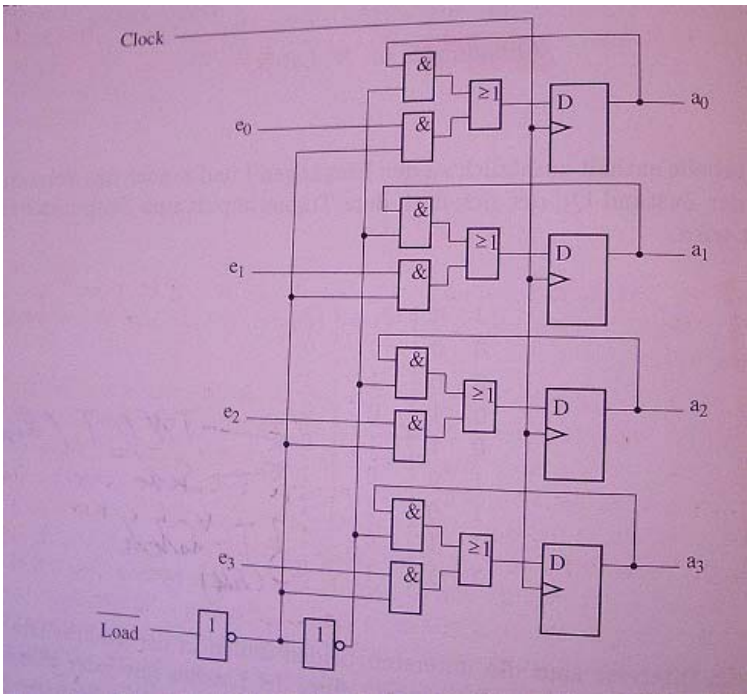
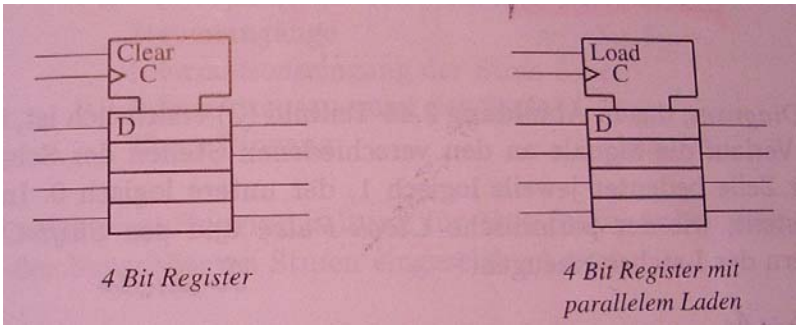


Abbildung 2.41: 4-Bit-Register

Schaltungen die es erlauben alle Bits eines Datenwortes simultan inner halb eines einzigen gemeinsamen Clock-Impuls aufzunehmen, nennt man Register mit parallelem Laden.

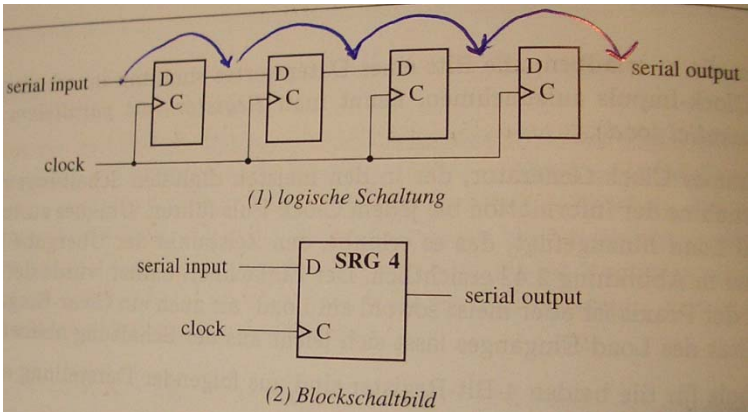


Blockschaltbilder von Registern:

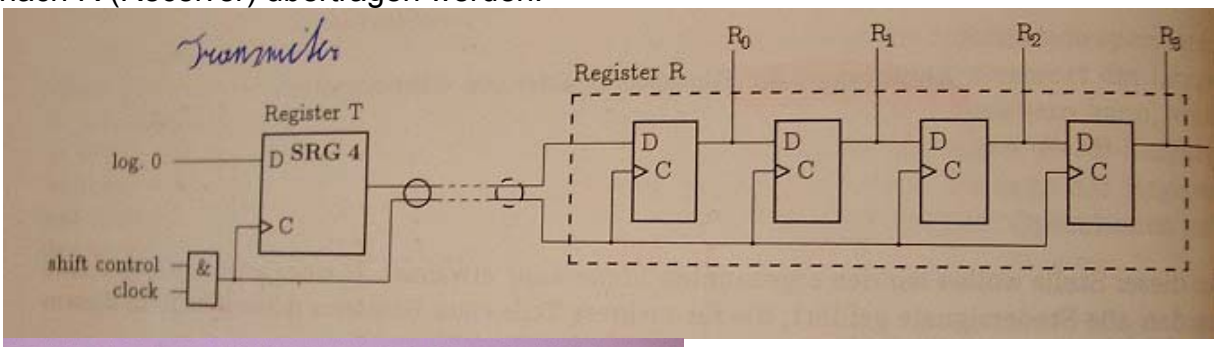


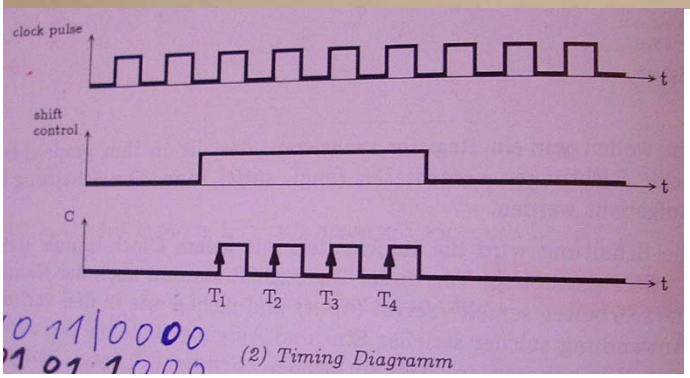
Schieberegister

Schieberegister sind Register die die gespeicherte Information in eine oder beide Richtungen verschieben kann, mit jedem Clock-Impuls wird die Information um genau eine Stufe (ein Latch) weiter geschoben. Der Eingang arbeitet nun Seriell und nicht wie bei den vorigen Registern parallel.



Durch die Schaltung in folgender Abbildung können Datenwörter von T (Transmitter) nach R (Receiver) übertragen werden:





Die folgenden Tabelle zeigt für die Werte T = 1011 und R = 0010 den Inhalt der Register und macht es möglich den Informationsfluss nachzuvollziehen.

timing pulse (C)	Register T				Register R			
					R_0	R_1	R_2	R_3
Anfangswerte	1	0	1	1	0	0	1	0
Nach T_1	0	1	0	1	1	0	0	1
Nach T_2	0	0	1	0	1	1	0	0
Nach T_3	0	0	0	1	0	1	1	0
Nach T_4	0	0	0	0	1	0	1	1

Die Ein- und Ausgänge in der nachfolgenden Abbildung haben folgende bedeutung:
S_0, S_1 Steuereingänge
e_i Informationseingang der Stufe i
a_i Informationseingang der Stufe i
clock Clock Eingang

Diese Stufe des Registers hat die Funktionen: parallel load, shift left, shift right

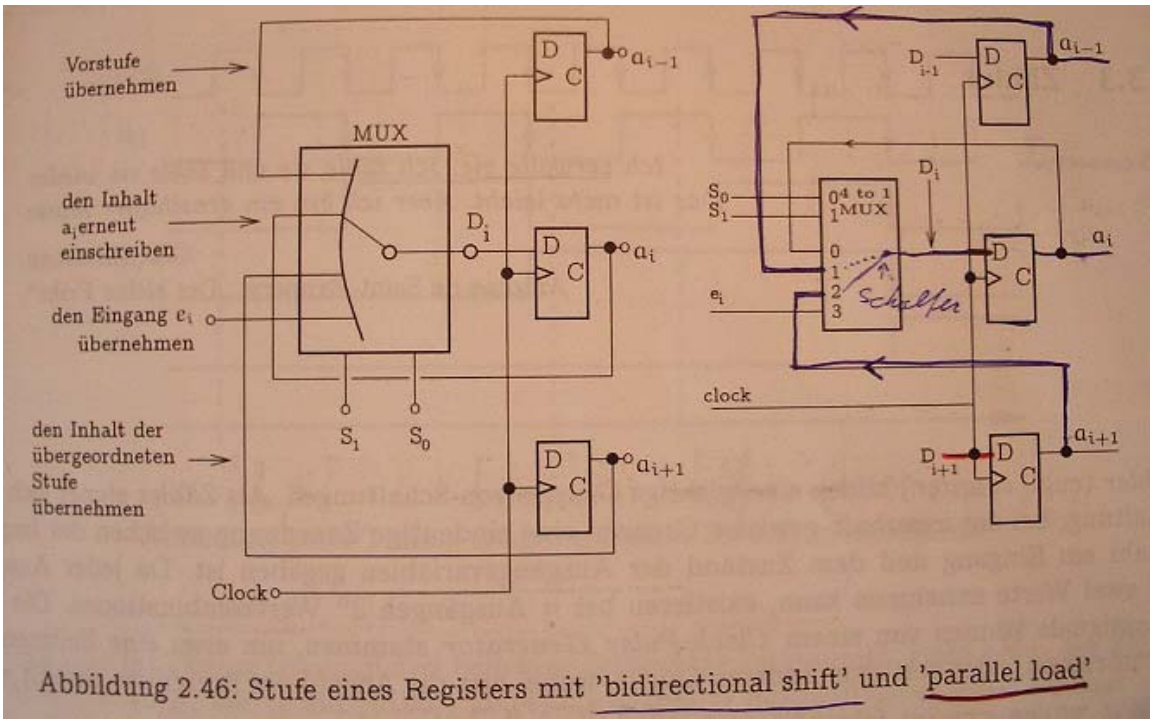
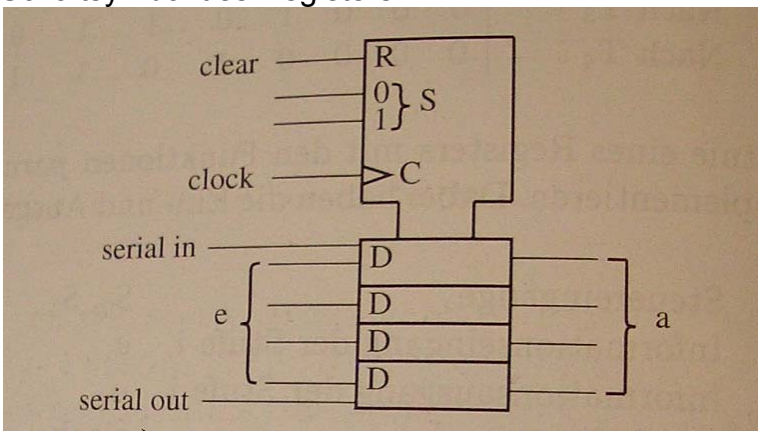


Abbildung 2.46: Stufe eines Registers mit 'bidirectional shift' und 'parallel load'

Es ergibt sich folgende Funktionstabelle:

control S_1	mode S_0	Register Operation
0	0	Keine Änderung
1	0	shift left (up)
0	1	shift right (down)
1	1	parallel load

Schaltsymbol des Registers:

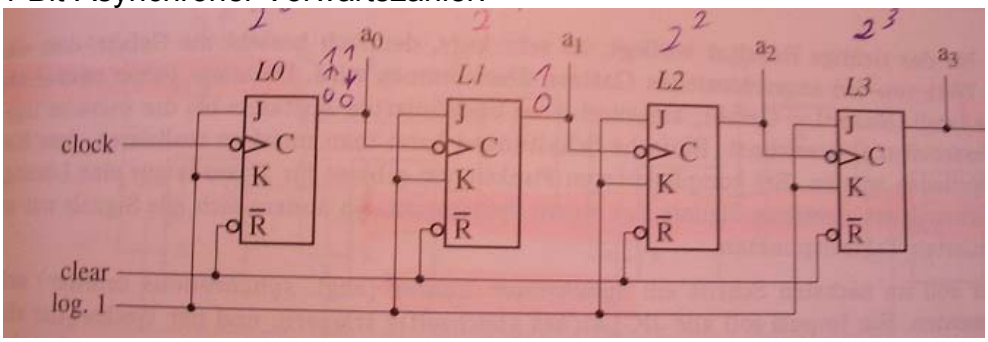


Zähler

Da ein Ausgang eines Zählers immer nur zwei Werte annehmen kann (0, 1) existieren bei n Ausgängen 2^n Wertekombinationen.
Man unterscheidet zwei Arten von Zählern, synchrone und asynchrone.

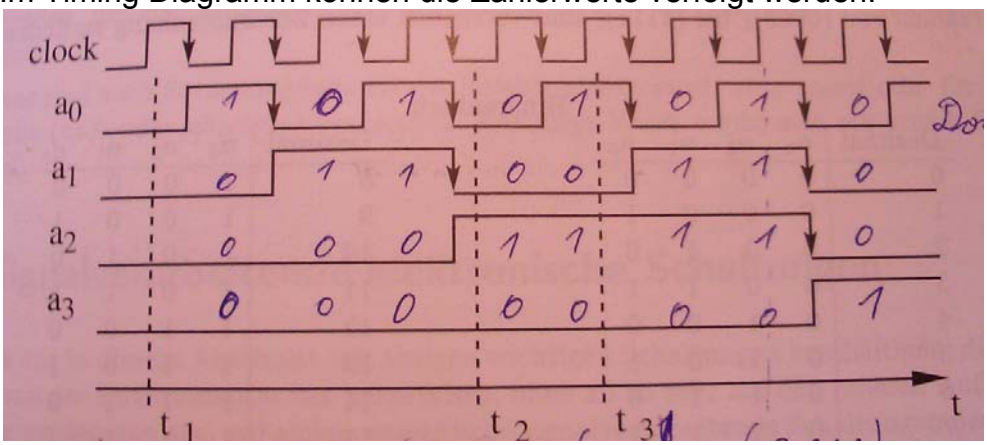
Asynchronzähler

4-Bit-Asynchroner Vorwärtszähler:



Beim Auftreten des ersten Impulses auf den Eingang C des ersten Latches L0 schaltet diese mit der fallenden Flanke um, und sein Ausgang a0 bekommt den Wert Logisch 1. Durch die nächste negative Flanke triggert L0 wieder, und a0 wechselt von logisch 1 auf logisch 0. Dies stellt wiederrum den Trigger für das Latch L1 dar, wodurch sein Ausgang a1 auf logisch 1 schaltet, usw.

Im Timing Diagramm können die Zählerwerte verfolgt werden:



Ein wesentlicher Nachteil eines Asynchronzählers liegt darin, dass außer dem ersten Latch alle weiteren nur indirekt angesteuert werden. Durch die Gatterlaufzeiten entstehen ungültige Zwischenwerte. Derartige Fehler werden als Hazards bezeichnet, um diese zu vermeiden synchronisiert man sämtliche Signale mit einem Systemtakt.

Synchronzähler

Für jede Stufe i müssen Funktionen für Ki und Ji (i = 0,1, …, n-1) ermittelt werden. Dazu sucht man für jeden möglichen Zustand $(an\text{-}1, an\text{-}2, \dots, a1, a0)_{alt}$ den richtigen Folgezustand $(an\text{-}1, an\text{-}2, \dots, a1, a0)_{neu}$. Für jede Zeile werden die Werte von Ki und Ji bestimmt.

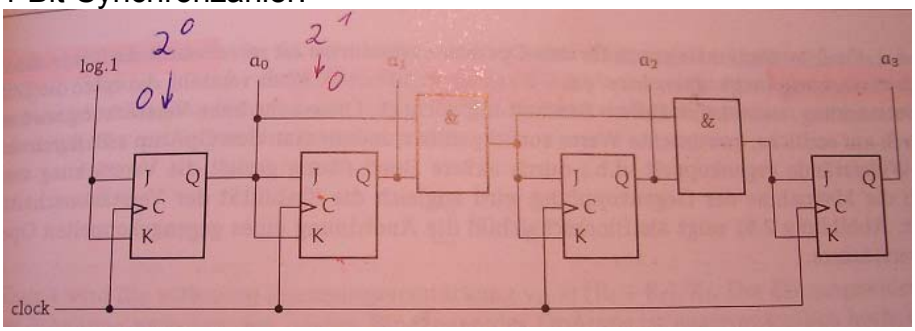
Binärzahlen

Dezimal	a_3	a_2	a_1	a_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1

Dezimal	a_3	a_2	a_1	a_0
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

a0 wechselt ständig seinen Zustand → J0 = K0 = 1
a1 wechselt immer ihren Wert wenn a0 log 1 ist → J1 = K1 = a0
a2 wechselt immer ihren Wert wenn die Vorgänger log 1 sind → J2 = K2 = a0 ^ a1
a3 analog zu a2 → J3 = K3 = a0 ^ a1 ^ a2

4-Bit-Synchronzähler:

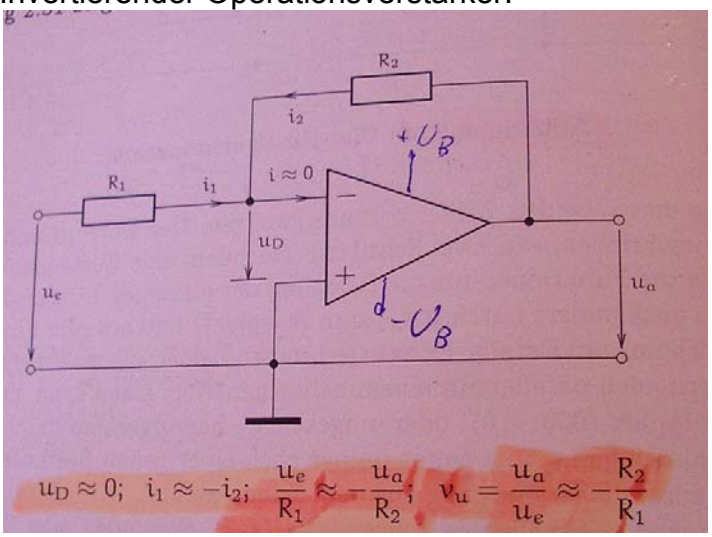


2.4 Signalverarbeitende elektronische Schaltungen

Operationsverstärker

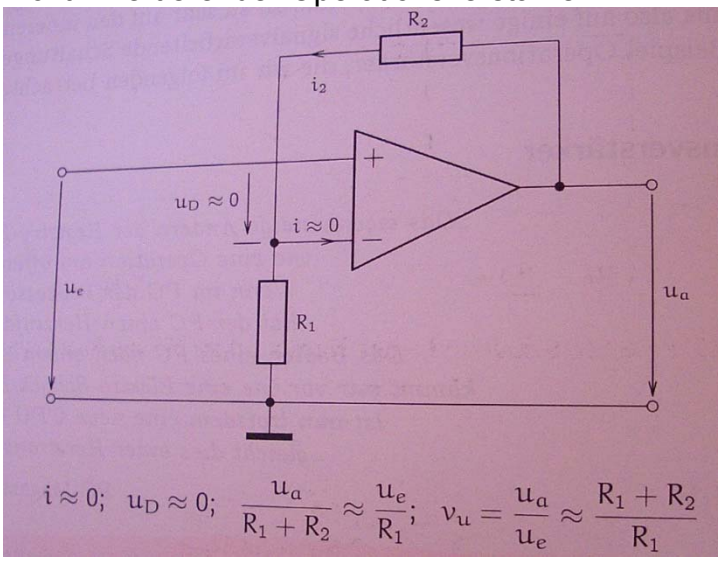
Operationsverstärker werden oft an der Schnittstelle zwischen analogen und digitalen Schaltkreisen benötigt.
Charakteristisch für den Operationsverstärker ist eine sehr hohe Leerlaufspannungsverstärkung $v_u = 10^5 \dots 10^8$

Invertierender Operationsverstärker:



Bsp: $v_u = u_a/u_e$ = 10V / 100 µV = 100000

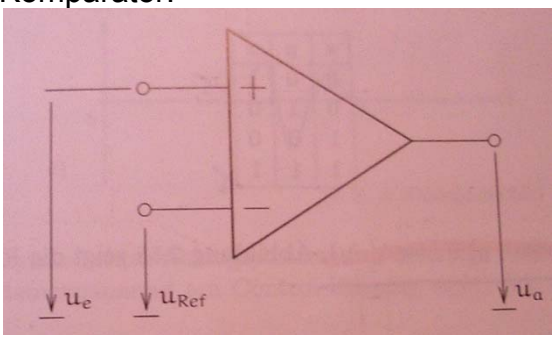
Nicht-Invertierender Operationsverstärker:



<u>Komparatoren</u>

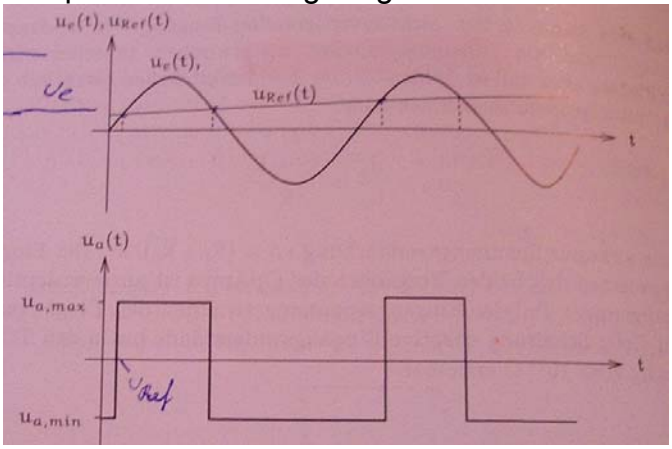
Sind Schaltungen mit denen festgestellt werden kann, ob und zu welchem Zeitpunkt die momentane Amplitude eines Signals gleich einem konstanten oder zeitlich veränderbaren Referenzsignal ist.

Komparator:



$u_a = U_{a\,max}$ für $u_e > U_{Ref}$
$u_a = U_{a\,min}$ für $u_e < U_{Ref}$

Komparator für Analoge Signale:



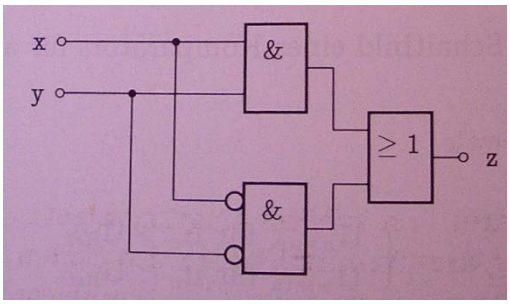
Komparator für Digitale Signale:

Die Arbeitsweise eines Komparators für digitale Signale kann durch die Äquivalenzfunktion beschrieben werden:

x ≡ y = z

x	y	z
0	0	1
0	1	0
1	0	0
1	1	1

z = (x ^ y) v (¬x ^ ¬y)

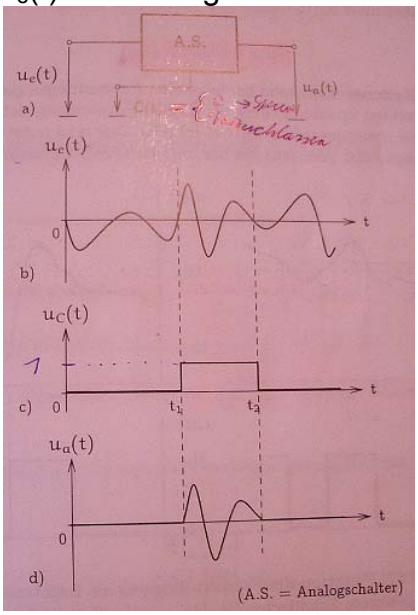


<u>Torschaltungen</u>

Unter Torschaltungen versteh man Netzwerke, die aus einem Signal für eine vorgegebene Zeit einen Zeitlichen Anteil des Signals „herausschneiden“.

Torschaltungen für analoge Signale:

Aus einem Analogen Signal lässt sich ein zeitlicher Ausschnitt des Eingangssignals $u_e(t)$ verzerrungsfrei an den Ausgang als $u_a()$ übertragen:



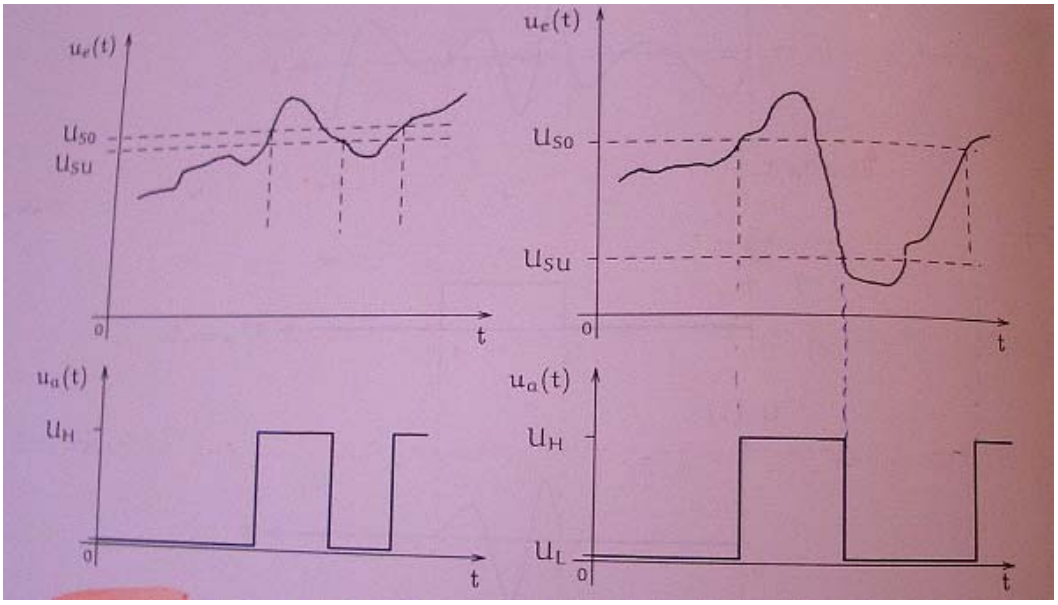
Toschaltungen für digitale Signale:

Übertragen ein binäres Eingangssignal im Idealfall verzerrungs- und verzögerungsfrei an den Ausgang zu den Zeiten, die durch das Control-Signal vorgegeben werden. Diese Schaltung wird einfach durch eine UND-Verknüpfung zwischen einer Variablen x und dem Control-Signal realisiert.

<u>Schmitt-Trigger</u>

Ist ein Komparator, bei dem Ein- und Ausschaltpegel um die Schalthysterese U_{HST} differieren. Überschreitet die Eingangsspannung $u_e(t)$ eine vorgegebene, obere Schaltschwelle U_{SO}, so nimmt der Ausgang der Schaltung einen binären Wert an, unterscheidet die Eingangsspannung die untere Schaltschwelle U_{SU}, so nimmt der Ausgang den anderen binären Wert an.

$U_{HST} = U_{SO} - U_{SU}$



Man sollte vom Einsatz eines Schmitt-Triggers in einer Umgebung die starken elektromagnetischen Störungen unterworfen ist absehen.

Invertierender Schmitt Trigger:

Die Ausgangsspannung u_a springt sehr schnell auf den Wert $U_{a\,min}$, wenn die Eingangsspannung u_e den Einschaltpegel erreicht bzw. überschreitet, andererseits springt die Ausgangsspannung u_a auf den Wert $U_{a\,max}$, wenn die Eingangsspannung u_e den Ausschaltpegel erreicht bzw. unterschreitet.

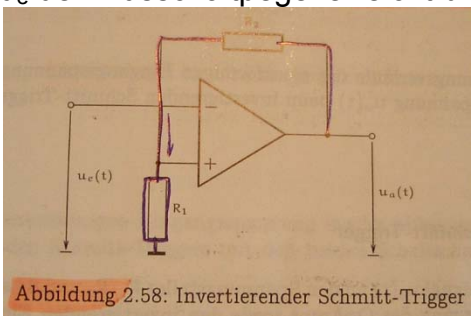
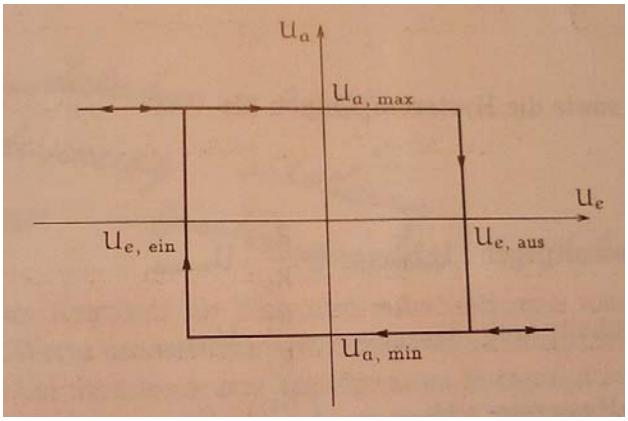
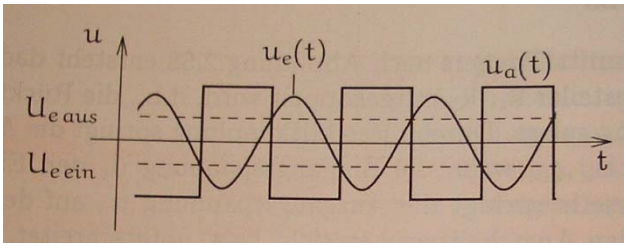


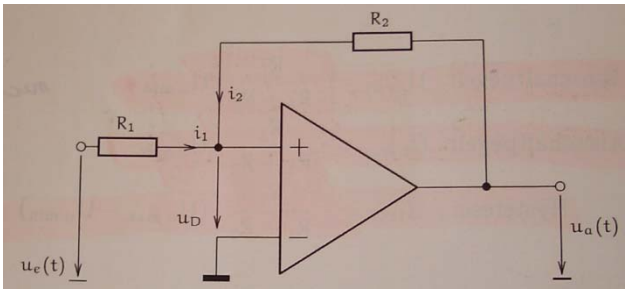
Abbildung 2.58: Invertierender Schmitt-Trigger

Einschaltpegel: $U_{e\ ein} = (R1/ (R1 + R2)) * U_{a\ min}$
Ausschaltpegel: $U_{e\ aus} = (R1/ (R1 + R2)) * U_{a\ max}$
Hysterese: $U_{HST} = (R1/ (R1 + R2)) * (U_{a\ max} – U_{a\ min})$

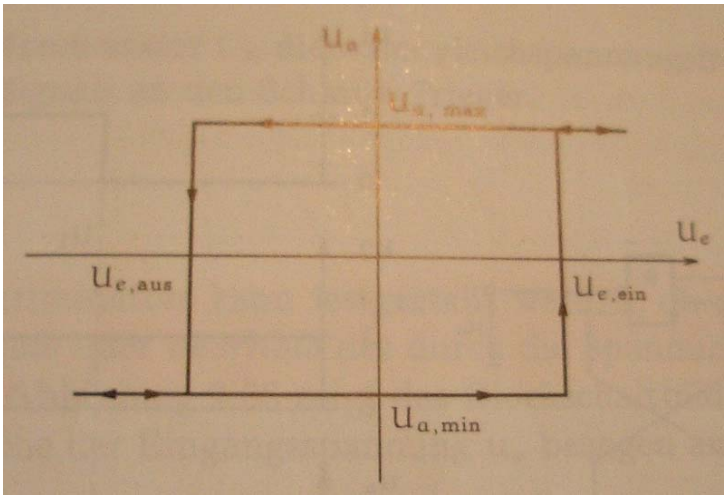


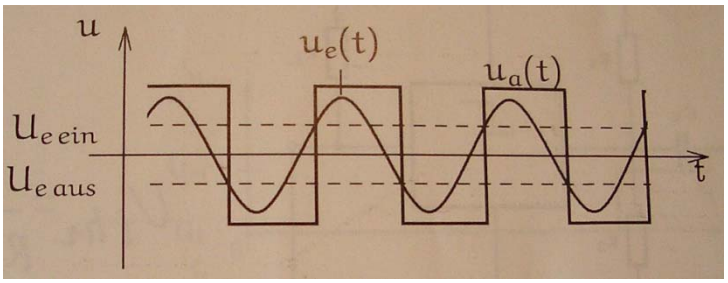


Nicht-Invertierender Schmitt Trigger:



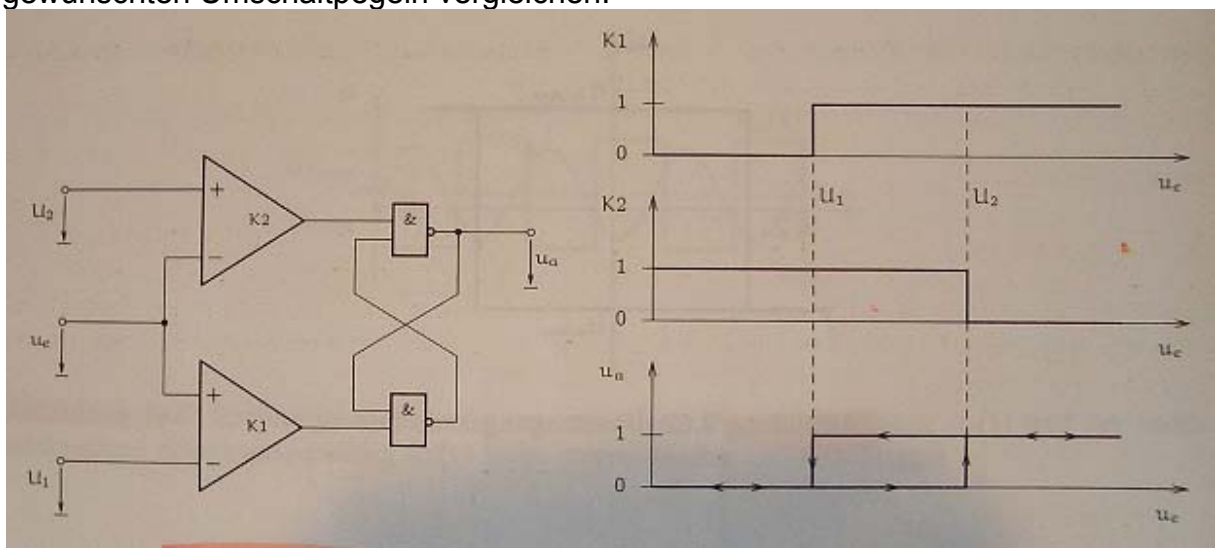
Einschaltpegel: $U_{e\ ein} = -(R1/R2) * U_{a\ min}$
Ausschaltpegel: $U_{e\ aus} = -(R1/R2) * U_{a\ max}$
Hysterese: $U_{HST} = (R1/R2) * (U_{a\ max} – U_{a\ min})$





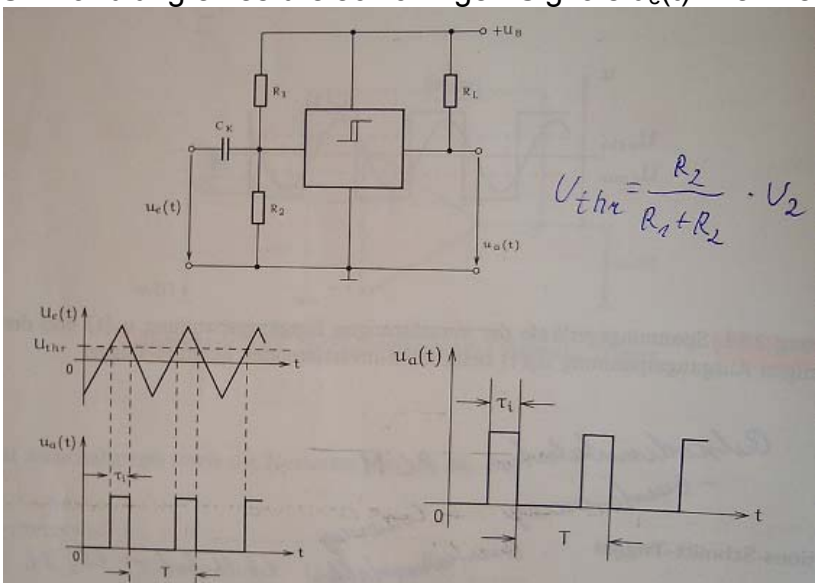
Präzisions-Schmitt-Trigger:

Ein- und Ausschaltpegel sowie die jeweilige Hysterese hängen entscheidend von den Werten $U_{a\,min}$ und $U_{a\,max}$ ab. Dieser Nachteil kann dadurch behoben werden in dem man 2 Komparatoren K1 und K2 verwendet, die das Eingangssignal mit den gewünschten Umschaltpegeln vergleichen.



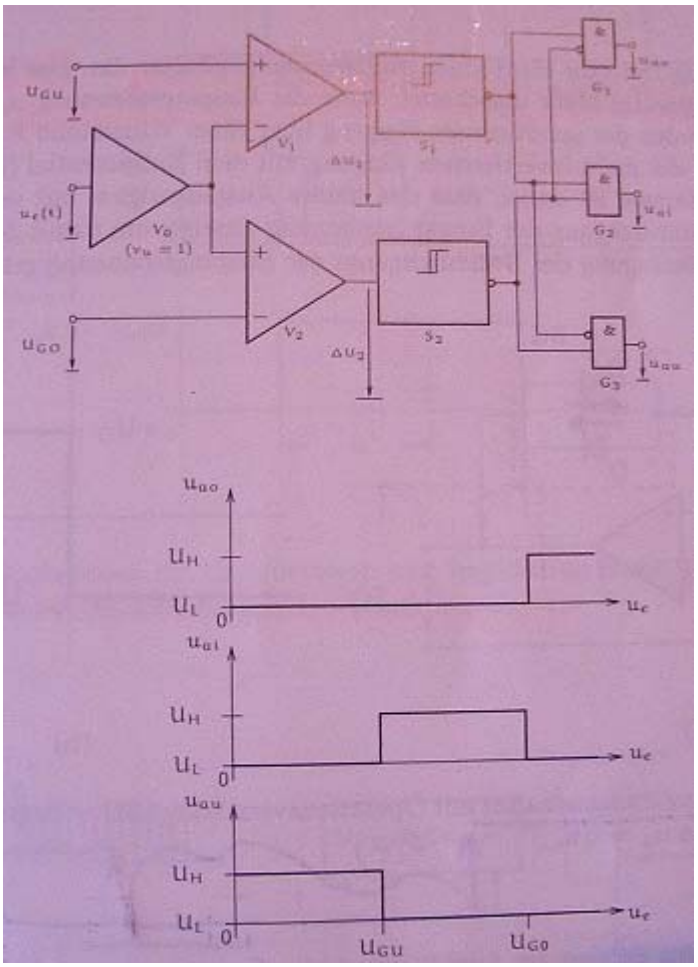
Einschaltpegel	$U_{e\,ein} = U_2$ mit $U_2 > U_1$
Ausschaltpegel	$U_{e\,aus} = U_1$

Umwandlung eines dreieckförmigen Signals $u_e(t)$ in ein rechteckiges Signal:



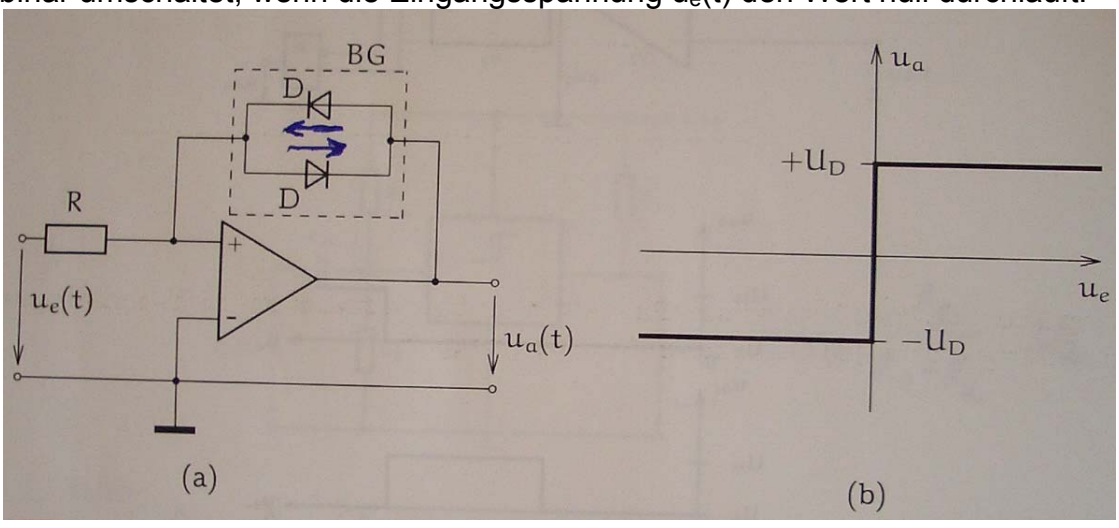
Fensterdiskriminator

Mit Hilfe eines Fensterdiskriminators kann festgestellt werden, ob der Wert der Eingangsspannung u_e unterhalb, innerhalb oder oberhalb des durch die Spannungsgrenzwerte U_{GO} und U_{GU} festgelegten Fensters liegt.



Zero-Crossing-Detector

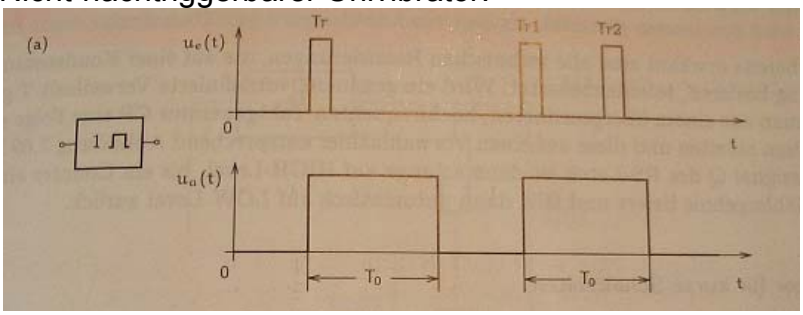
Ist ein Nullspannungsschalter, das bedeutet, dass dieser Schalter ausgangsseitig binär umschaltet, wenn die Eingangsspannung $u_e(t)$ den Wert null durchläuft.



Univibrator

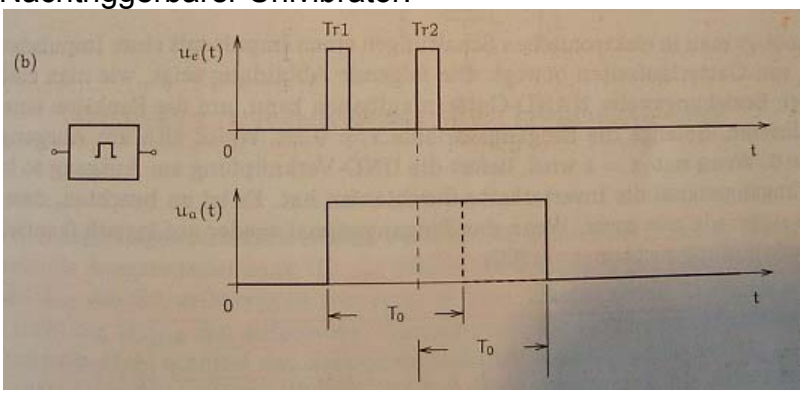
Stellt eine Schaltung dar die ausgangsseitig zwei Zustände annehmen kann. Von diesen beiden Zuständen ist nur einer stabil, der andere ist quasi-stabil und kann für eine vordefinierte Zeit, die Verweilzeit T0, nach einem eingangsseitigen Triggerimpuls angenommen werden. Diese Schaltungsanordnung wird auch als monostabile Kippstufe bezeichnet.

Nicht-nachtriggerbarer Univibrator:



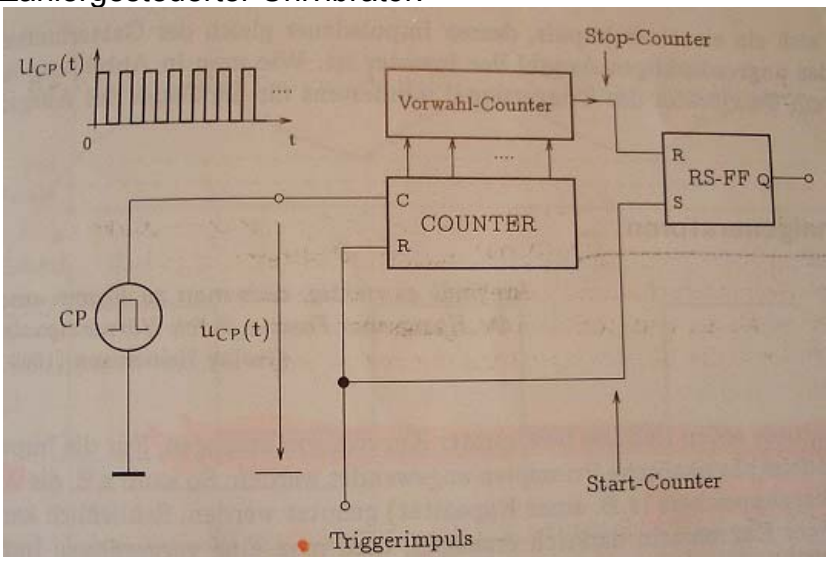
Trifft ein weiterer Triggerimpuls vor Ablauf der Verweilzeit am Eingang ein, dieser aber unberücksichtigt bleibt, so liegt ein nicht-nachtriggerbarer Univibrator vor.

Nachtriggerbarer Univibrator:



Trifft ein weiterer Triggerimpuls vor Ablauf der Verweilzeit am Eingang ein, und die Verweilzeit wird von neuem gestartet, so liegt ein nachtriggerbarer Univibrator vor.

Zählergesteuerter Univibrator:

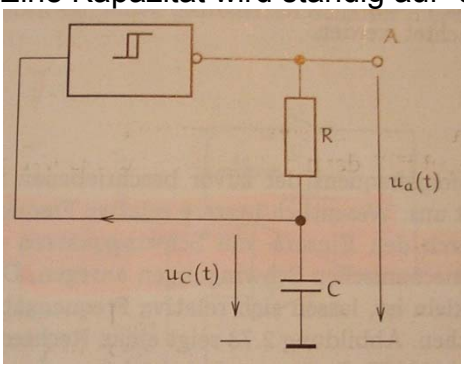


Signalgeneratoren

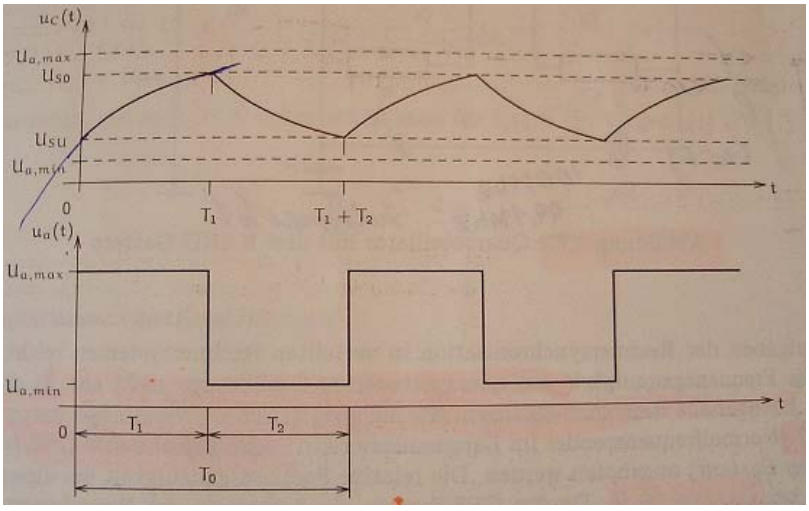
Signalgeneratoren sollen Impulse bestimmter Kurvenform erzeugen. Signale können durch Auf- und Entladung eines Energiespeichers (z.B. einer Kapazität), durch Beschreibung einer Impulszeitfunktion durch endlich viele Stützstellen und Interpolation zwischen den Ordinaten, oder durch Schwingquarze erzeugt werden.

Rechteckgeneratoren:

Eine Kapazität wird ständig auf- und entladen.

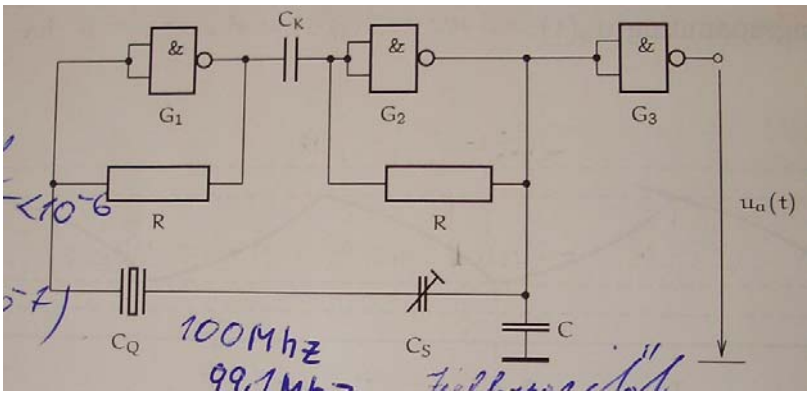


Der Schwellwertschalter schaltet am Ausgang auf maximale Ausgangsspannung $U_{a\,max}$, sobald die Spannung am Kondensator die untere Schaltschwelle U_{su} des Schwellwertschalters erreicht bzw. überschreitet. Dadurch wird die Kapazität C in Richtung $U_{a\,max}$ hin aufgeladen. Überschreitet die Kondensatorspannung $u_c(t)$ die obere Schaltschwelle U_{so}, schaltet der Schwellwertschalter ausgangsseitig auf $U_{a\,min}$ zurück, so dass die Kapazität C wieder entladen wird, bis die Kondensatorspannung die untere Schalschwelle U_{su} unterschreitet. Dieser Auf- und Entladevorgang wiederholt sich somit periodisch.



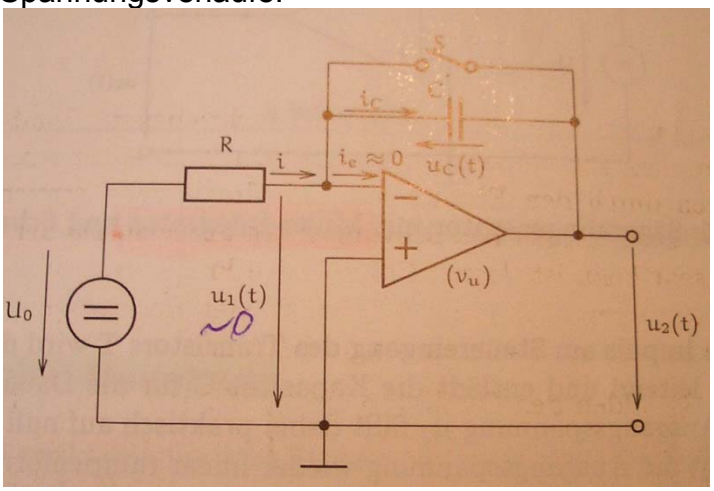
Quarzoszillator

Wesentlich bessere realtive Frequenzkonstanz Δf/f kann man bei Rechteckgeneratoren durch den Einsatz von Schwingquarzen erreichen, diese werden durch elektrische Felder zu mechanischen Schwingungen angeregt.



Sägenzahngenerator

Der Miller-Integrator eignet sich besonders zur Erzeugung sägezahnförmiger Spannungsverläufe.

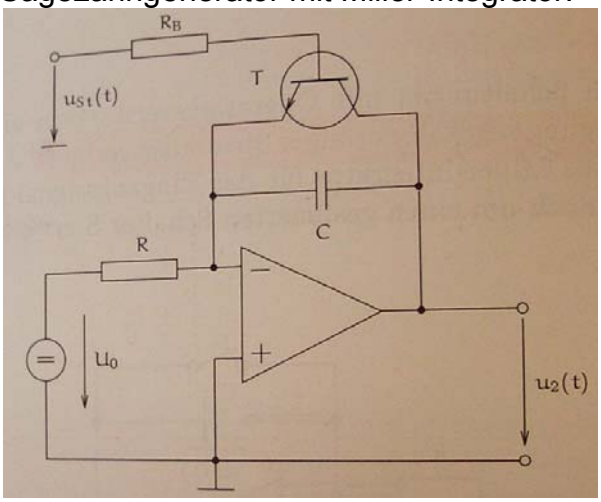


Durch periodisches Schließen des Schalters S und die dadurch entstehende Entladung der Kapazität C entsteht ein sägezahnförmiger Verlauf.

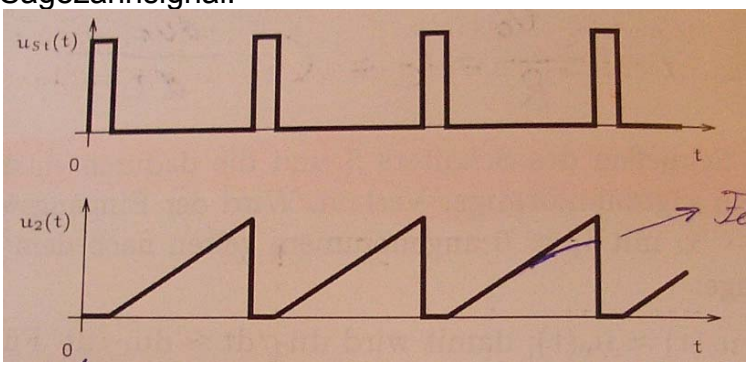
$I_C(t) = C * du_C /dt$

$u_2(t) = U_0/(R*C) * t$

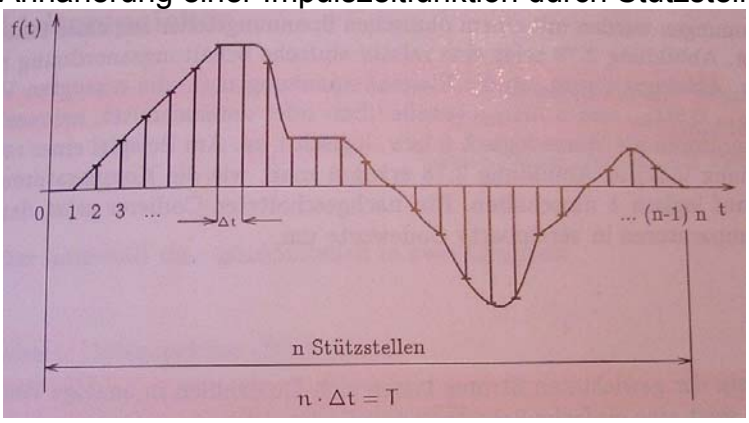
Sägezahngenerator mit Miller-Integrator:



Sägezahnsignal:



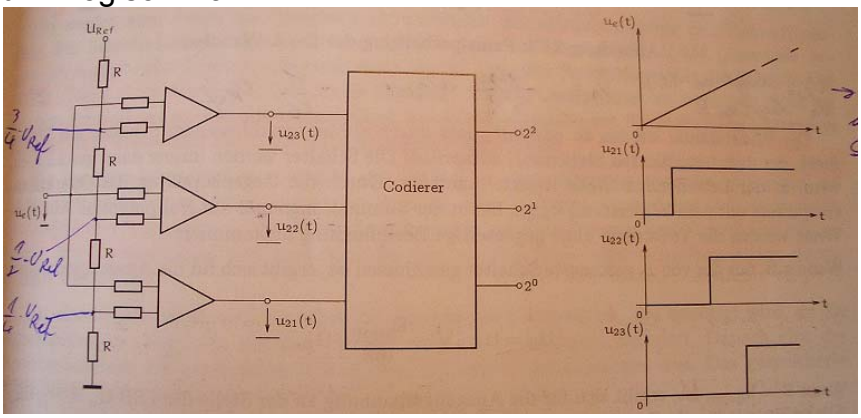
Annäherung einer Impulszeitfunktion durch Stützstellen:



Analog-Digital Umsetzer

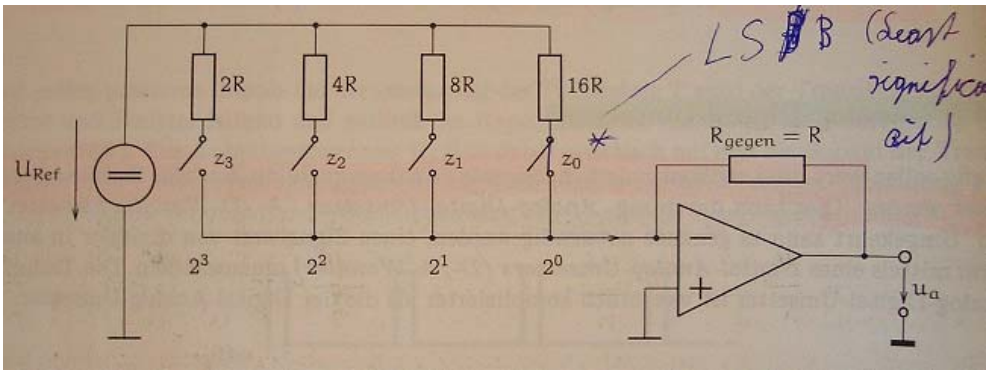
Häufig müssen Analoge Signale Digital weiterverarbeitet werden, deshalb werden A-D-Wandler oft benötigt.

Ein entsprechende Anzahl von Komparatoren wird parallel angeordnet und mit der Eingangsspannung $u_e(t)$ beschaltet. Die Vergleichsspannungen werden aus einer Referenzspannung abgeleitet. Abhängig davon, ob die Eingangsspannung $u_e(t)$ dje erzeugten Vergleichsspannungen $0{,}25U_{Ref}$, $0{,}5U_{Ref}$ und $0{,}75U_{Ref}$ jeweils über oder unterschreitet, nehmen die Ausgänge der drei Komparatoren die Werte logisch 0 bzw. logisch 1 an.



Digital-Analog Umsetzer

Nach dem Prinzip der gewichteten Ströme lassen sich Dualzahlen in analoge Werte umwandeln.



Die Widerstände werden so gewählt, dass durch sie bei geschlossenem Schalter ein Strom fließt, der dem betreffenden Stellenwert entspricht.

$U_a = U_{LSB} = - (R_{gegen}/16R) * U_{Ref}$

$U_a = -1/2*U_{Ref}z3 -1/4*U_{Ref}z2 -1/8*U_{Ref}z1 -1/16*U_{Ref}z0$

2.5 Halbleiterspeicher

Halbleiterspeicher unterteil man in 2 Gruppen:

Tabellenspeicher: Datenspeicher
Funktionsspeicher: Speicherung von logischen Funktionen

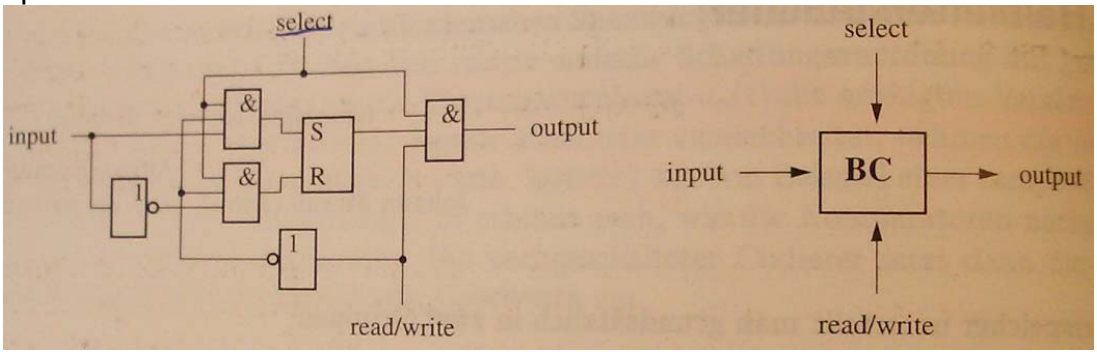
Tabellenspeicher

8 Bit = 1 Byte

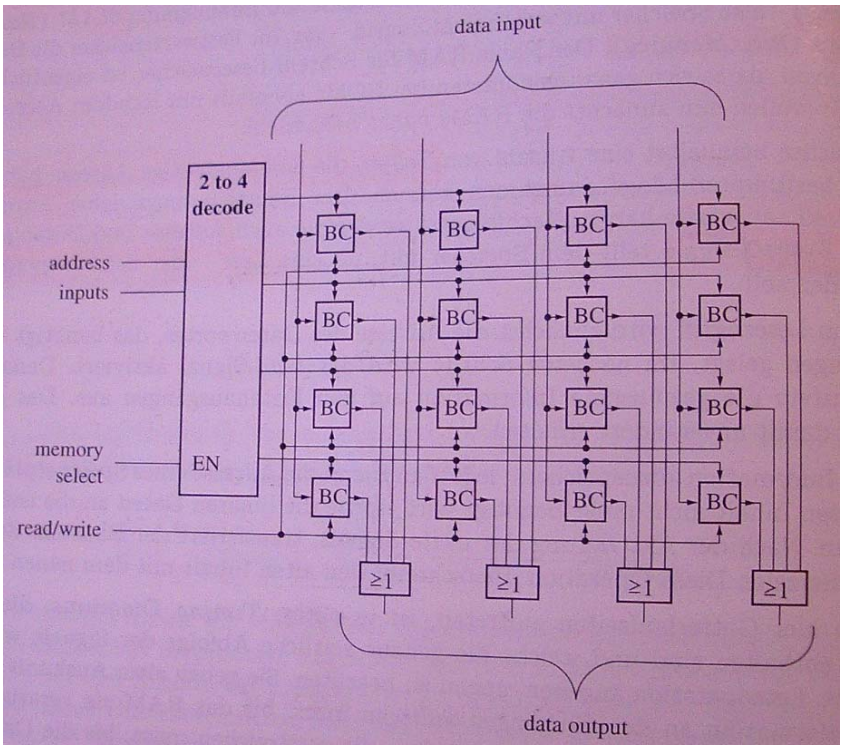
Es wir unterschieden zwischen:
Schreib-Lesespeicher (RAM = Random Access Memory) und
Festwertspeicher (ROM = Read Only Memory)

Ein speicher beinhaltet Zellen die eine eindeutige Adresse haben und ein Datenwort bestimmter Länge aufnehmen können.
Ein solcher Baustein hat deshalb Adresseingänge, Datenein- und –ausgänge und einen Kontrolleingang um anzugeben ob eine Lese- oder Schreiboperation erfolgt.

Speicherzelle:



Um zu vermeiden das ständig Informationen am Datenausgang anstehen, z.B. 00, wird ein Enable Eingang EN eingeführt. Wenn EN=0 liegen keine Daten am Ausgang.



4x4 Bit, die erste Ziffer steht für die Anzahl der Wörter die gespeichert werden können und die zweite Ziffer steht für die Länge eines Wortes.
Mit n Adressleitungen können 2^n Wörter adressiert werden.

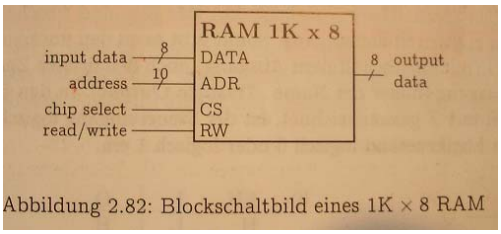


Abbildung 2.82: Blockschaltbild eines 1K × 8 RAM

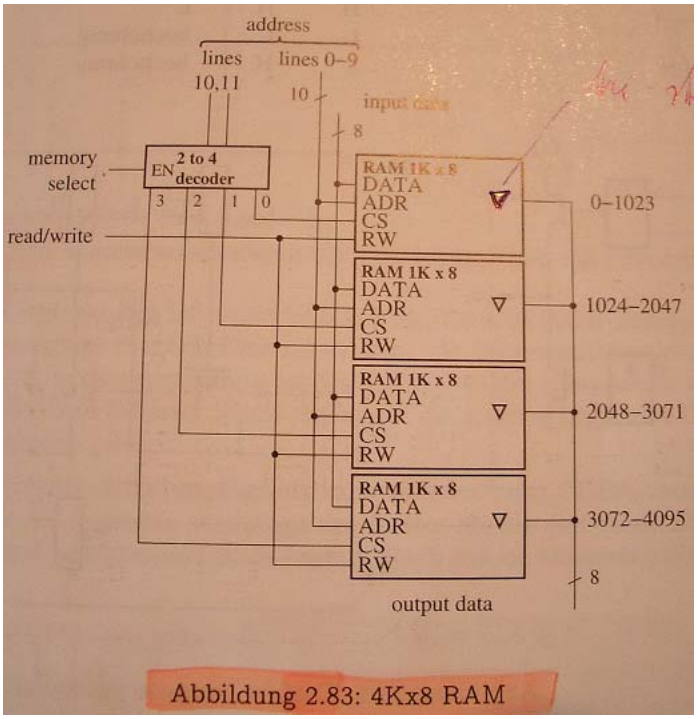
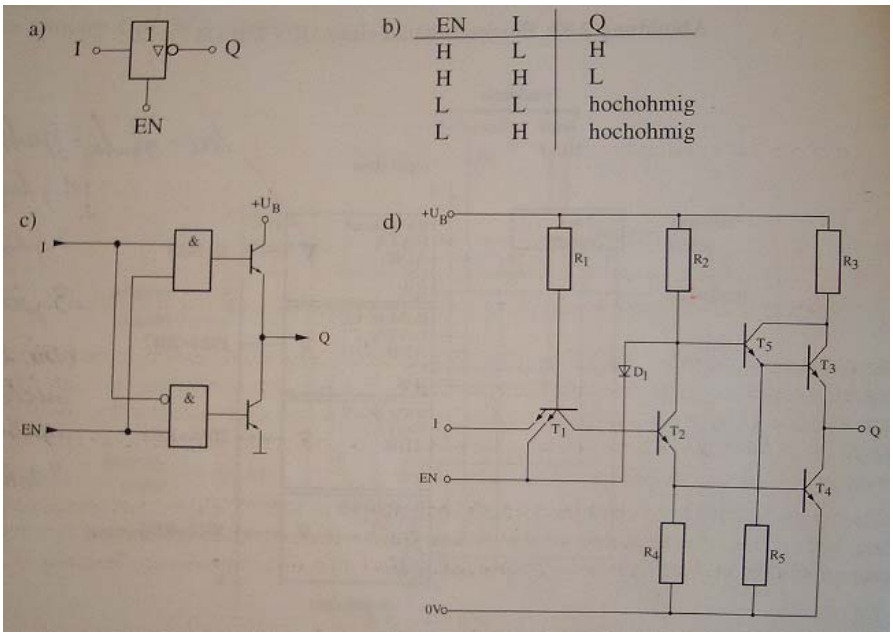


Abbildung 2.83: 4Kx8 RAM

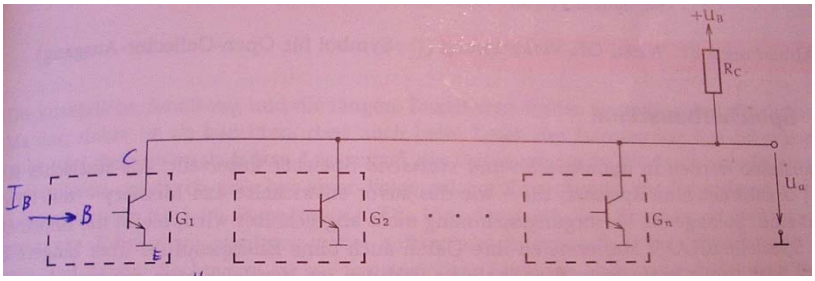
Tristate Outputs

Oft ist es notwendig mehrere Ausgänge zusammenzuschalten, dies kann jedoch zu Problemen führen. Tristate Outputs stellen abschaltbare Ausgänge dar. Das Ein- und Abschalten des Ausganges erfolgt dabei über einen eigenen Steuereingang.



Open-Collector-Schaltungen

Manchmal müssen viele Gatter ausgangsseitig miteinander verknüpft werden. Dies lässt sich bewältigen in dem man Gatter mit offenem Kollektor Ausgang (Open-Collector) einsetzt.



Der Ausgang u_a befindet sich bei positiver Logik nur dann im HIGH Zustand, wenn alle angeschlossenen Gatterausgänge ebenfalls HIGH sind. Sobald sich auch nur ein Ausgang im LOW Zustand befindet geht die Ausgangsspannung auch in den LOW Zustand. Somit ergibt sich ein UND Verknüpfung.
Ein ODER Verknüpfung ergibt sich indem man die negierten Ausgänge der Gatter mit ihrer offenen Kollektoren zusammenschaltet und anschließend noch negiert.

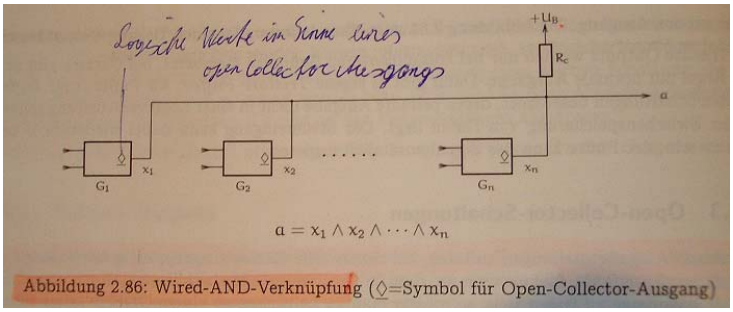


Abbildung 2.86: Wired-AND-Verknüpfung (◊=Symbol für Open-Collector-Ausgang)

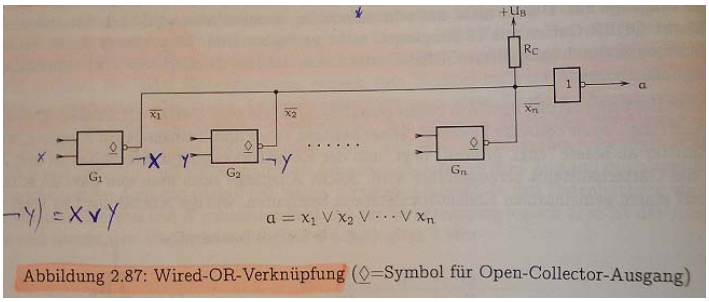
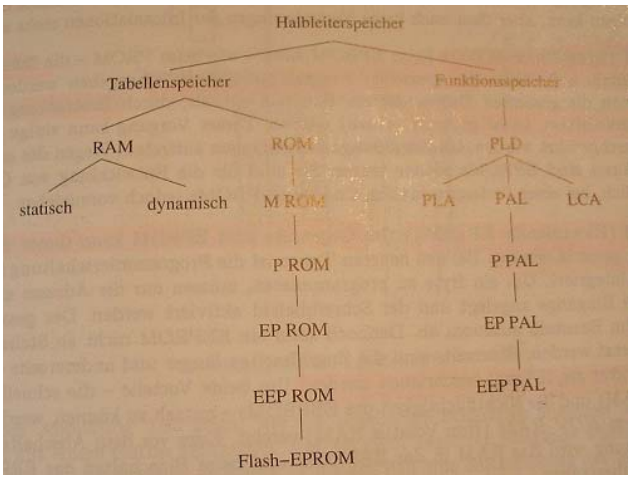


Abbildung 2.87: Wired-OR-Verknüpfung (◊=Symbol für Open-Collector-Ausgang)

Speicherbausteine

RAMs werden in dynamisch und statisch Speicher unterteilt.
Statische RAMs (SRAM) sind solche RAMs wie zuvor vorgestellt.
Dynamische RAMs (DRAM) speicher die Information nicht in Latches sondern in Kondensatoren. DRAMs müssen alle 8 ms einen Refresh Cycle durchführen, bei dem die gespeicherte Information jeder Speicherzelle ausgelesen und neu eingeschrieben wird.

Burst Refresh: Refresh bei allen Speicherzellen, kein Zugriff auf den Speicher möglich
Cycle Stealing: Refreshvorgänge für einzelne Teile des Speichers getrennt durchführen
Transparent Refresh: Refresh Controller wird mit dem Prozessor synchronisiert, so dass laufende Prozesse nicht angehalten werden müssen



Von ROMs können nur Daten ausgelesen werden, der Inhalt ist nicht flüchtig und bleibt auch ohne Stromzufuhr erhalten.

ROM (Read Only Memory): Inhalt wird bei der Herstellung definiert und kann nicht mehr verändert werden.
PROM (Programmable ROM): Der Inhalt kann vom Anwender mit einer besonderen Schaltung einprogrammiert werden, jedoch nur einmal.
EPROM (Erasable PROM): Information kann vom Anwender eingegeben und durch Bestrahlung der ICs mit ultraviolettem Licht wieder gelöscht werden.
EEPROM (Electrically EPROM): Baustein kann beschrieben und elektrisch gelöscht werden, jedoch nicht öfter als 10^6 mal (langsamer als RAMs)
Flash-EPROM: Nur der ganz Chip ist elektrisch löschbar, nicht bitweise

Funktionsspeicher (ASICs)

ASIC = Application Specific Integrated Circuit, sind Bauteil die zur Speicherung einer Funktion dienen, die gesamte Schaltung ist hier in ein Bauteil integriert.
Die wichtigsten Vertreter sind: PALs, PLAs, LCAs, PLDs, und Gate Arrays

PAL = Programmable Array Logic
PLA = Programmable Logic Array

Beide enthalten AND und OR Gatter. Die Eingangsvariablen bilden mi den kreuzenden Eingängen der UND-Gatter eine Matrix, mit welcher man alle benötigten Konjunktionen herstellen kann.

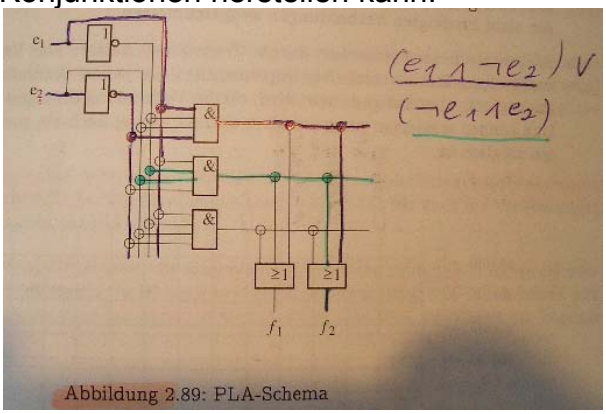


Abbildung 2.89: PLA-Schema

P**LA**s können durch folgende Verfahren programmiert werden:

1. Einmal programmierbar nach dem Prinzip der Schmelzsicherung, die Verbindungen werden durch dünne Halbleiterstriefen hergestellt, die durch einen Stromimpuls durchgeschmolzen werden.
2. Mehrmals programmierbar durch Transistorschalter, die Verbindungen werden durch einen Transistor implementiert.

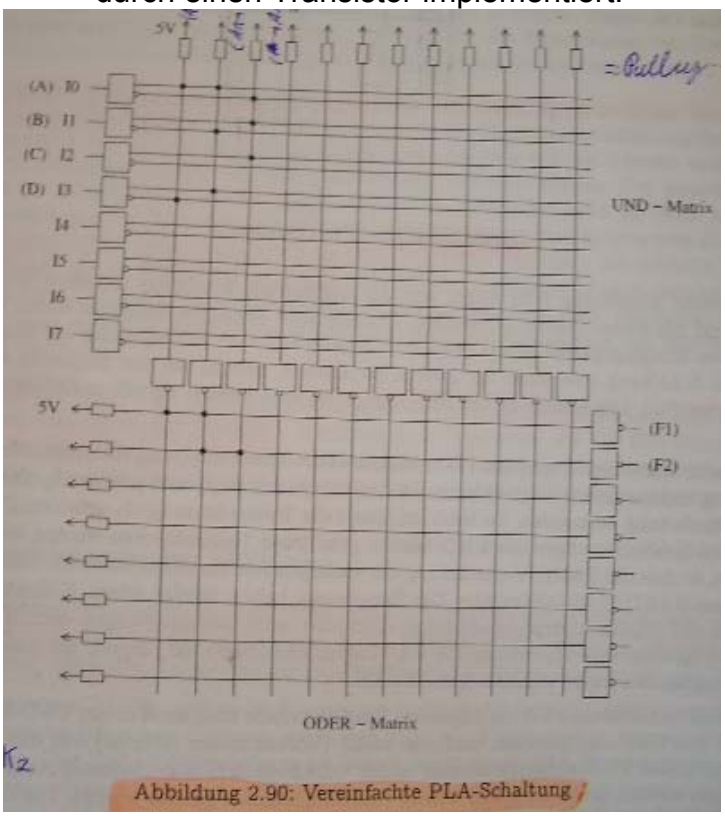


Abbildung 2.90: Vereinfachte PLA-Schaltung

LCAs (Logic Cell Array), sind Logikbausteine die aus einer Matrix von PALs bestehen.
PLDs (Programmable Logic Device), sind dem Aufbau eines PLAs ähnlich, haben mehrere Kombinationsmöglichkeiten und verfügen auch über Latches.

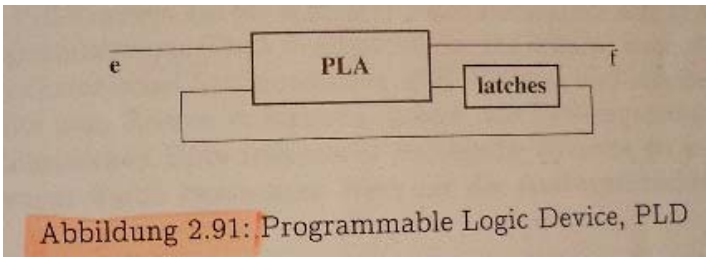


Abbildung 2.91: Programmable Logic Device, PLD

PLAs und PLDs lassen sich lediglich nur einmal programmieren.

Gate Arrays lassen sich nicht vom Anwender direkt Programmieren, der Kunde entwirft aber mit Hilfe von CAD und CAE den gewünschten IC und der Halbleiterhersteller passt die Funktion der Standardchips individuell an.

3.VHDL

VHDL = Very (High Speed Integrated Circuit) Hardware Description Language
VHDL ist eine Sprache für Hardwaredesign.

Entwurfssichten

Es wird von drei Entwurfssichten ausgegeangen:

Geometrie: Wie sieht die Komponente aus?
Verhalten: Was macht die Komponente?
Struktur: Woraus ist die Komponente aufgebaut?

Diese Entwurfssichten sind im Y-Diagramm dargestellt:

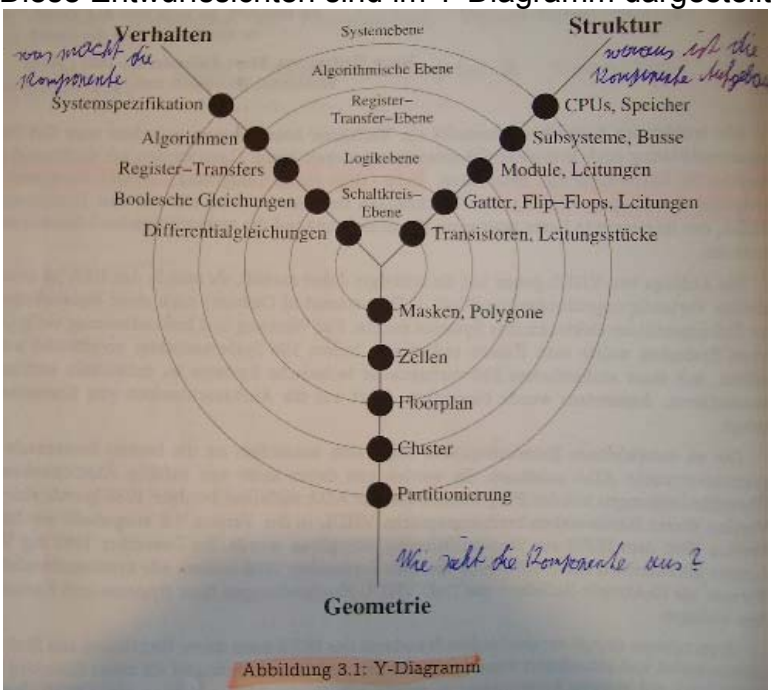


Abbildung 3.1: Y-Diagramm

Ein Designprozess geht von außen zum Mittelpunkt hin, man kann sich den Entwurf elektrischer Systeme als eine Reihe von Transformationen und Verfeinerungen im Y-Diagramm vorstellen, je weiter man zu Mittelpunkt kommt, desto mehr Details werden sichtbar.

Entwurfsebenen

Systemebene

Aus der Verhaltenssicht werden die grundlegenden Charakteristika des elektronischen Systems beschrieben. Aus der Sicht der Struktur wird die Grundfunktionalität in Blöcke eingeteilt. Unterteilung der Chipfläche aus der Sicht der Geometrie.

Algorithmische Ebene

Aus der Verhaltenssicht wird die Schaltung durch eine Algorithmische Darstellung mit Variablen und Operatoren beschrieben. Die Strukturelle Sicht liefert für die Schaltung eine Beschreibung durch Blöcke die durch Signale miteinander kommunizieren. Aus der Geometrie Sicht werden Cluster definiert.

Register-Transfer-Ebene

Aus der Verhaltenssicht wird die Schaltung durch Operationen und den Datentransfer beschrieben. Aus Struktureller Sicht werden Register, Codierer und ähnliche Komponenten durch Signale miteinander Verknüpft. Aus der Sicht der Geometrie werden die Chipflächen grob in einen Flurplan eingeteilt.

Logikebene

Aus der Verhaltenssicht werden zur Beschreibung der Schaltung Boolsche Ausdrücke und Wahrheitstabellen eingesetzt.
Die Struktur wird hier durch Gatter(Gedächtnislos), Latches (Gedächtnis behaftet) und Leitungen beschrieben.
Die Geometrie liefert Zellen als Bestandteil des Flurplans.

In Bauteilbibliotheken werden alle wesentlichen elektrischen Eigenschaften der verwendbaren Bauelemente durch Kennwerte hinterlegt, sodass mit Simulatoren das sequenzielle Schaltverhalten simuliert werden kann.

Schaltkreisebene

Bezüglich des Verhaltens werden Differenzialgleichungen zur Modellierung des Systemverhaltens herangezogen. Aus der Sicht der Struktur werden die elektrischen Bauelemente zu einer sogenannten „Netzliste“ zusammengefasst. Aus der Sicht der Geometrie werden Polygonzüge zur Darstellung verwendet.

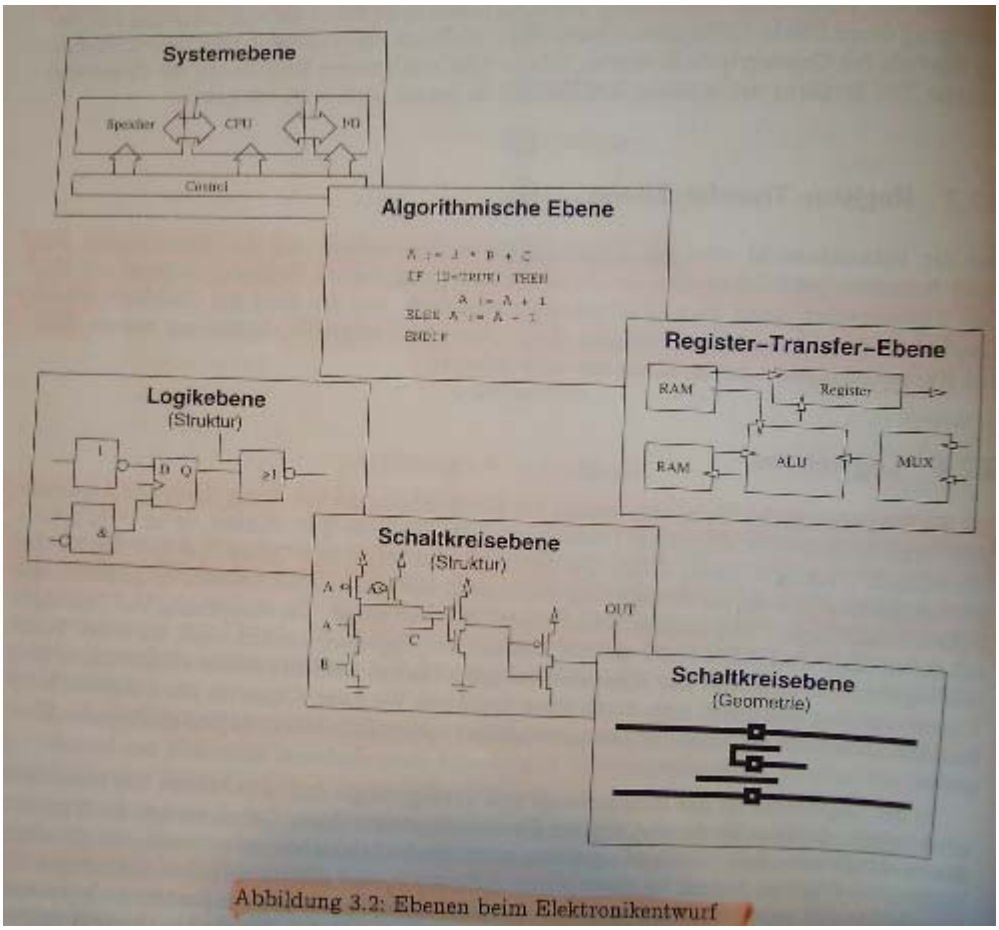


Abbildung 3.2: Ebenen beim Elektronikentwurf

Während auf den oberen Ebenen Systeme hoher Komplexität gut beherrschbar sind, liefern die unteren Ebenen mehr Details bzw. höhere Genauigkeit.

Systemebene und algorithmische Ebene unterstützen die Dokumentation des Gesamtsystems, Register-Transfer-Ebene und Logik-Ebene ermöglichen Simulationen.

Der Aufbau einer VHDL-Beschreibung

Es sind drei Bearbeitungsschritte erforderlich:

Schnittstellenbeschreibung
Architektur
Konfiguration

Schnittstellenbeschreibung

Ein- und Ausgänge, Konstanten, Unterprogramme und sonstige Vereinbarungen werden hier definiert.

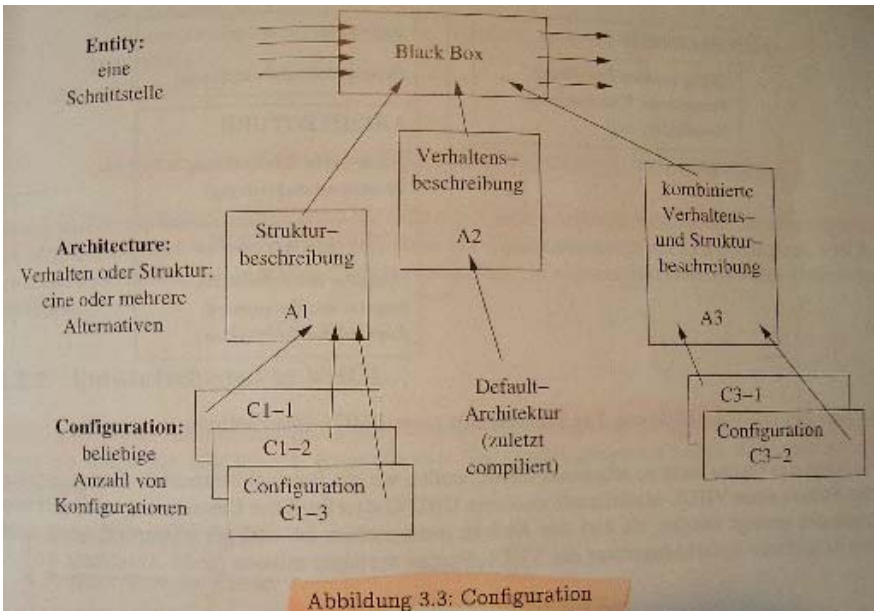


Abbildung 3.3: Configuration

entity entity_name **is**
    port (in/out connections)
end entity_name;

Architektur

Beschreibt die Funktionalität des Moduls/Komponente entweder als Verhaltensbeschreibung oder als Netzliste.

architecture architecture_name **of** entity_name **is**
    [local variables]
begin
    [processes]
end architecture_name;

Konfiguration

In der Konfiguration wird Festgelegt, welche Architektur einem Modul/Konfiguration zugeordnet wird.

process_name: **process** (sensitivity_list)
begin
    [Anweisungsblock]
end process process_name;

Arrays: vname: **vector**(a **to** b) oder vname: **vector**(a **downto** b)
Zugriff auf Element: vname(index) := 0;
Auf ganzes Feld: vname := „<Werte-Liste>“
An Variablen: x := 0
An Signale: out_signal <= 0

Packages: **library** IEEE;
use IEEE.std_logic_1164.all;

Bestandteile einer VHDL-Beschreibung

Üblicherweise enthält ein VHDL-Tool bereits ein sogenanntes Package, d.h., Anweisungen für Typ- und Objektdeklarationen und wie man Prozeduren und Funktionen beschreibt.

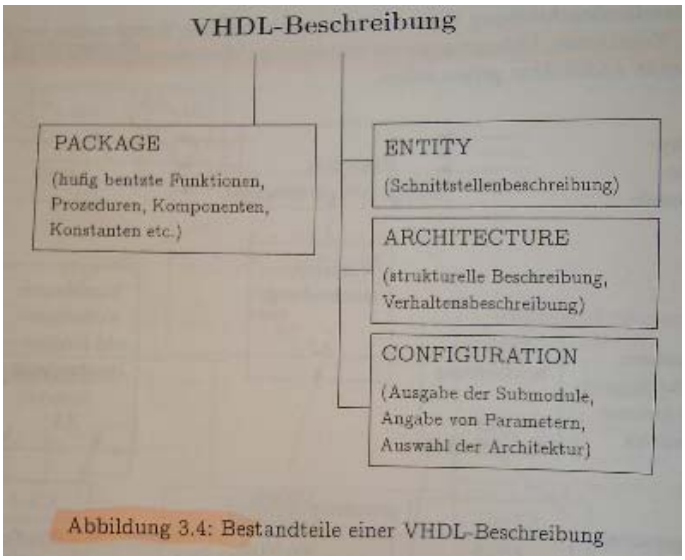
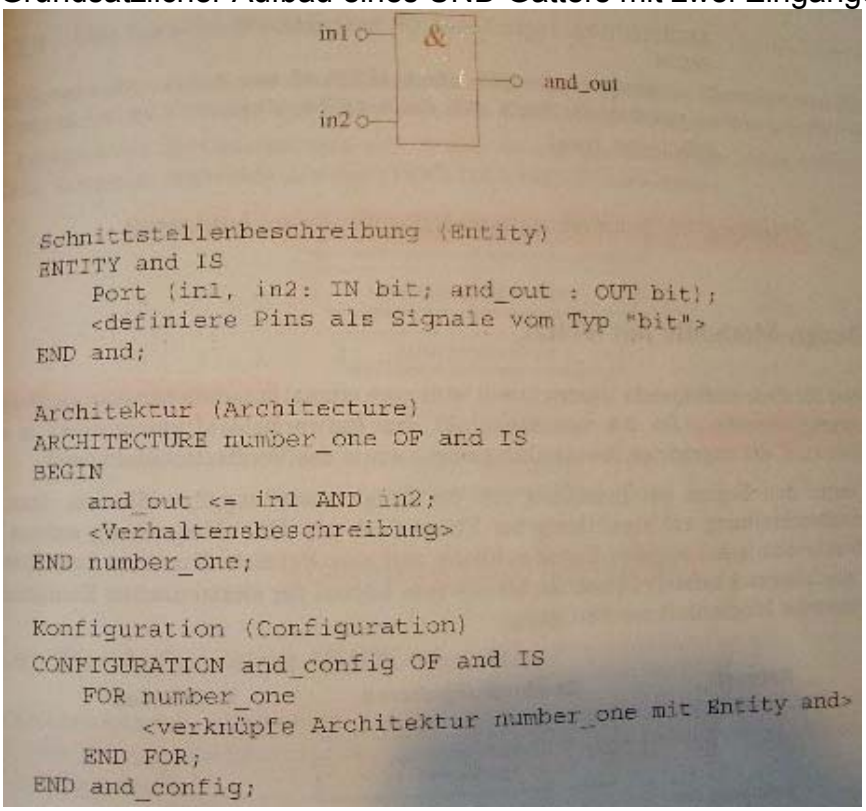


Abbildung 3.4: Bestandteile einer VHDL-Beschreibung

Entwurfssichten in VHDL

VHDL ermöglicht eine Verhaltensmodellierung sowie eine Strukturelle Modellierung. Bei der Verhaltensmodellierung wird das Verhalten einer Komponente durch die Reaktion der Ausgangssignale auf Änderungen der Eingangssignale beschrieben. Bei der strukturellen Modellierung werden die Eigenschaften eines Modells durch seinen inneren Aufbau aus Unterkomponenten dargestellt.

Grundsätzlicher Aufbau eines UND Gatters mit zwei Eingängen in VHDL




```
Schnittstellenbeschreibung (Entity)
ENTITY and IS
    Port (in1, in2: IN bit; and_out : OUT bit);
    <definiere Pins als Signale vom Typ "bit">
END and;

Architektur (Architecture)
ARCHITECTURE number_one OF and IS
BEGIN
    and_out <= in1 AND in2;
    <Verhaltensbeschreibung>
END number_one;

Konfiguration (Configuration)
CONFIGURATION and_config OF and IS
    FOR number_one
        <verknüpfe Architektur number_one mit Entity and>
    END FOR;
END and_config;
```

Entwurfsebenen in VHDL

VHDL unterstützt Beschreibungen in verschiedenen Entwurfsebenen, ausgehend von der Systemebene bis hinab zur Logikebene. Folgende Beschreibungsebenen haben die größte Bedeutung:

Algorithmische Ebene
Register-Transfer-Ebene
Logikebene

Auf der Logikebene werden für ein elektronisches System die logischen Verknüpfungen digitaler Signale und deren zeitliche Eigenschaften beschrieben.

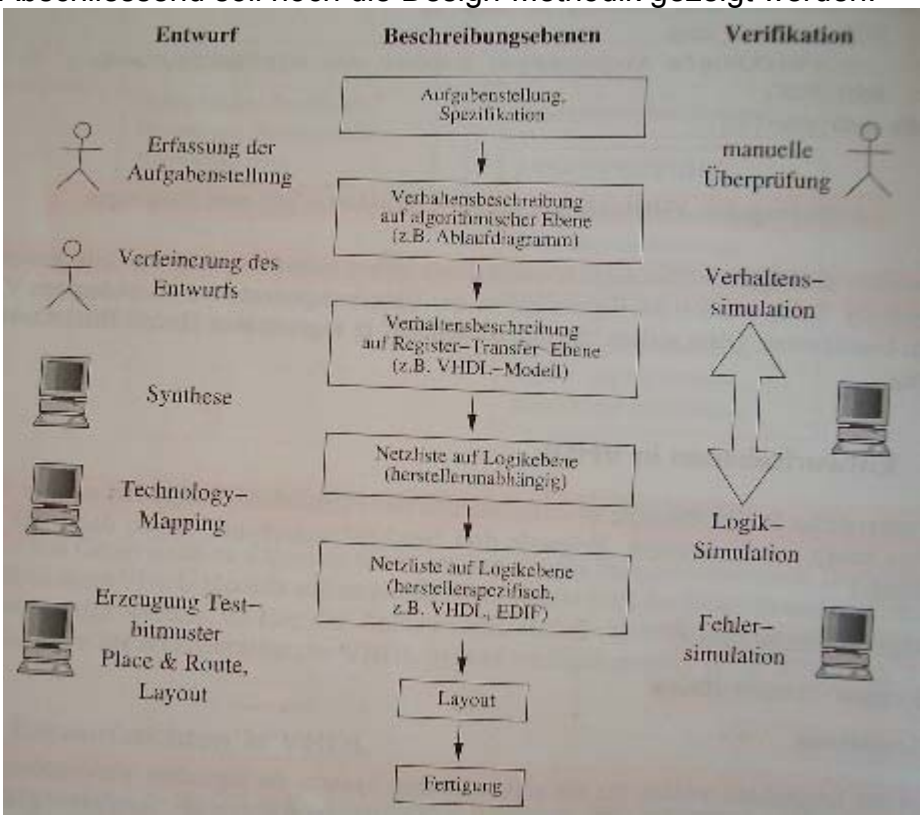
Beschreibung eines Halbaddierers auf Logikebene:

```
ARCHITECTURE logic_level OF halfadder IS
BEGIN
    sum <= sum_a XOR sum_b AFTER 15 ns;
    carry <= sum_a AND sum_b AFTER 10ns;
END logic_level
```

Abbildung 3.6: Beschreibung eines Halbaddierers auf Logikebene

Design-Methodik mit VHDL

Abschliessend soll noch die Design-Methodik gezeigt werden:



Die Sprache VHDL

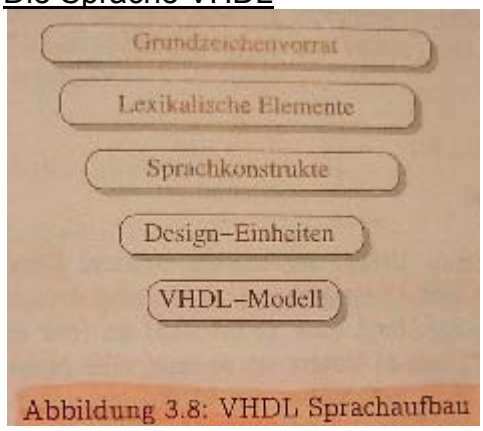


Abbildung 3.8: VHDL Sprachaufbau

Der Aufbau eines VHDL-Modells

VHDL besteht aus einer Schnittstellenbeschreibung einer oder mehreren Verhaltens- oder Strukturbeschreibungen und Konfigurationen.

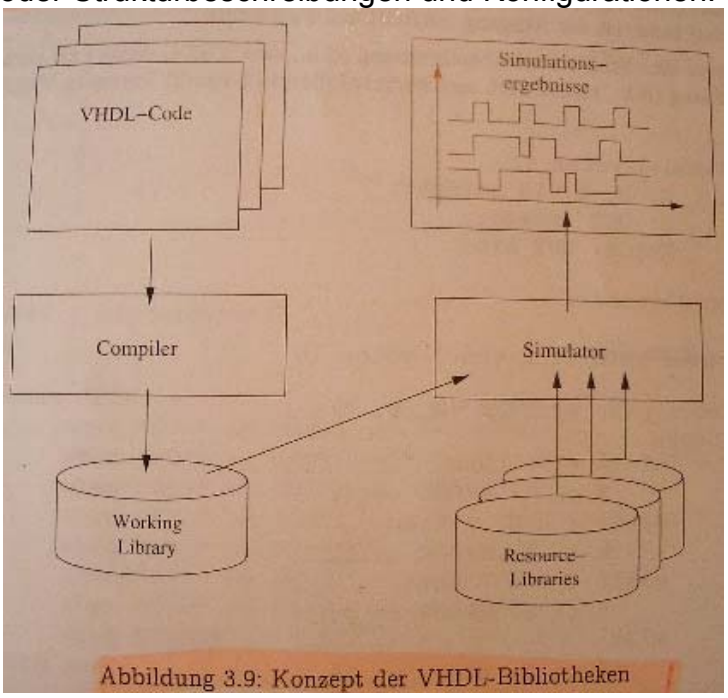


Abbildung 3.9: Konzept der VHDL-Bibliotheken

Ein strukturell aufgebautes VHDL-Modell greift auf hierarchisch gegliederte VHDL-Beschreibungen zu, indem ein Konzept mit Bibliotheken angewandt wird. Diese Bibliotheken dienen als Aufbewahrungsort für compilierte und wieder zu verwendende Design-Einheiten.

Beispiele

2-von-3 Voter

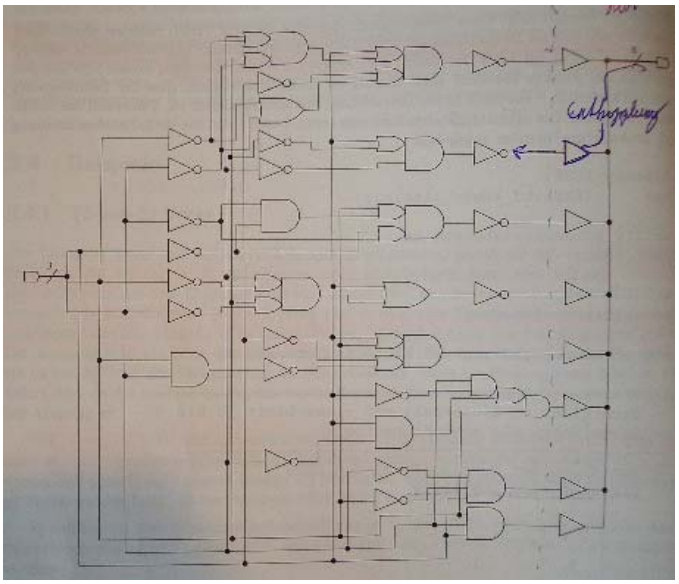
Es gibt drei Eingänge A, B und C und einen Ausgang R. Die Aufgabe eines solchen Voters ist es eine Mehrheitsentscheidung zu treffen und jenen Wert am Ausgang R zu liefern, der am häufigsten unter den drei Eingängen auftritt. Ausserdem gibt es noch einen ERROR Ausgang der dann auf 1 gesetzt wird, wenn alle drei Eingänge unterschiedlich sind.

```
entity 2-von-3-voter is
    port (  A, B, C: IN integer;
            R: OUT integer;
            ERROR: OUT bit;
    )
end 2-von-3-voter;

architecture vote of 2-von-3-voter is

        vote_proc process (A, B, C)
        begin
            if (A = B) then
                R <= A; ERROR <= 0;
            elsif ( A=C  ) then
                R <= A; ERROR <= 0;
            elsif ( B=C ) then
                R <= B; ERROR <= 0;
            else
                R <= 0; ERROR <= 1;
        end if;
    end vote_proc;
end vote;
```

Siebensegment-Decoder

Eine 3-Bit-Binärzahl soll als Dezimalzahl auf einer Siebensegment-Anzeige sichtbar gemacht werden. Der Input ist damit ein 3-Bit-Datenwort, der Output ein 8-Bit-Datenwort.

```
library IEEE;
use     IEEE.std_logic_1164.all;

entity binaertosiebenseg is
port (data: in std_logic_vector(2 downto 0);
      digit: out std_logic_vector(7 downto 0)

);
end binaertosiebenseg;

architecture behaviour of binaertosiebenseg is

-------------------------------------------------------
-- Darstellen v. Binaerzahl auf einem Digit (0 bis 7)
-- Ausgefuehrt als asynchrone Logik
-------------------------------------------------------
-- Zuordnung der Bussignale zu den LEDs
-- Die LEDs sind LOW-Aktiv
--      __5__
--     |     |
--     |4    |0
--     |__6__|
--     |     |
--     |3    |1
--     |__2__|  .7
-------------------------------------------------------

begin
    CONVERT : process(data)
    begin

        case data is                          76543210
            when "000" => digit <= "01000000";
            when "001" => digit <= "01111100";
            when "010" => digit <= "00010010";
            when "011" => digit <= "00011000";
            when "100" => digit <= "00101100";
            when "101" => digit <= "00001001";
            when "110" => digit <= "00000001";
            when OTHERS => digit <= "11011100";
        end case;
    end process CONVERT;
end behaviour;
```

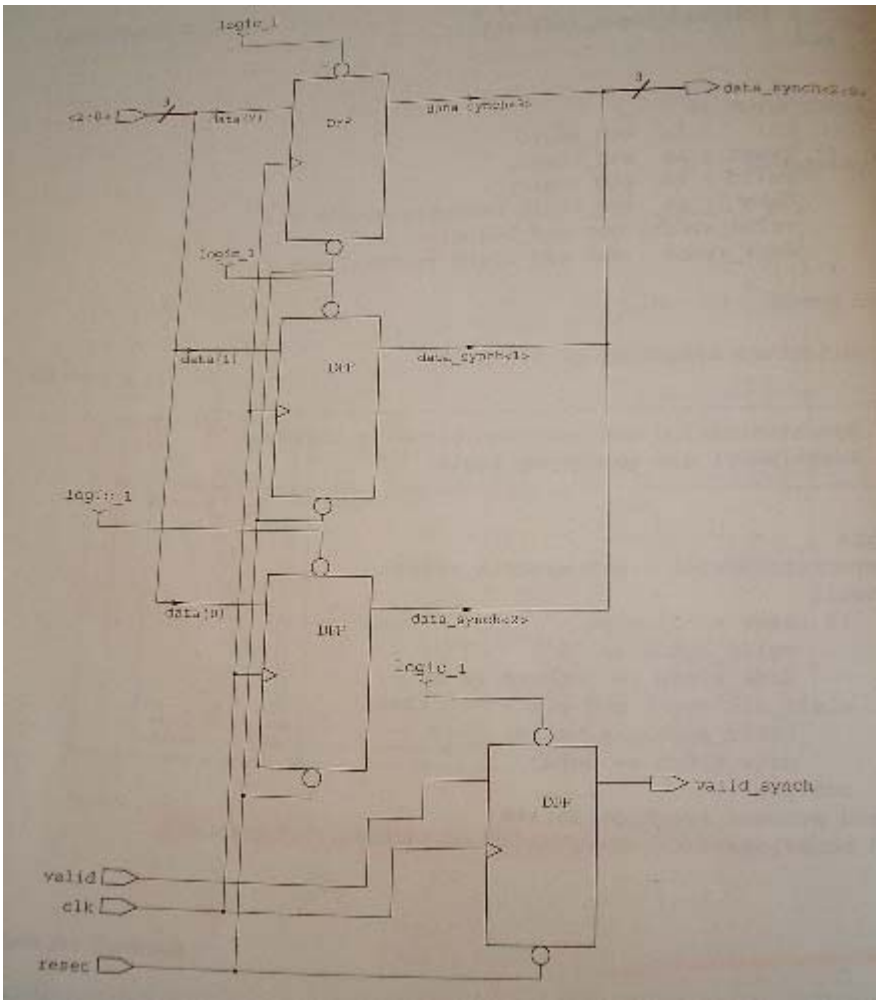


Schaltbild des Siebensegment-Decoders (US-Norm)

Input-Synchronisation

Ein asynchrones Eingangssignal soll über einen Eingangs-Latch synchronisiert werden. Im vorliegenden Beispiel wird ein 3 Bit breiter Bus über je einen Latch synchronisiert.

```
library IEEE;
use     IEEE.std_logic_1164.all;


entity synch is
 port (clk    : in  std_logic;
       reset  : in  std_logic;
       valid  : in  std_logic;
       data   : in  std_logic_vector(2 downto 0);
       valid_synch: out std_logic;
       data_synch : out std_logic_vector(2 downto
       );
end synch;

architecture behaviour of synch is

-----------------------------------------
-- Synchronisation von externem Signal & Datenbus
-- Ausgefuehrt als synchrone Logik
-----------------------------------------

begin
  synchronisation : process(clk,reset)
  begin
    if reset = '0' then
        valid_synch <= '0';
        data_synch <= (others => '0');
    elsif clk'event and clk = '1' then
        valid_synch <= valid;
        data_synch <= data;
    end if;
  end process synchronisation;
end behaviour;
```



Schaltung der Input-Synchronisation (US-Norm)

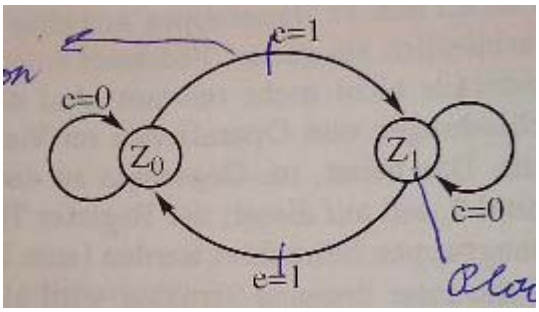
4.Mikorprozessoren

Logische Schaltungen sollen nun zu Baugruppen mit einer komplexeren Funktionalität zusammengesetzt werden und schließlich ein ganzer Prozessor entstehen.
(Register-Transfer-Ebene)

Endliche Automaten

Automat: ist ein System, das verschiedene Zustände annehmen kann.
Endlicher Automat: die Anzahl der Zustände ist Endlich
Deterministischer Automat: aus der Eingangsinformation und dem Vorzustand des Automaten lässt sich stets eindeutig der Folgezustand angeben
Endlicher deterministischer Automat: besitzt sowohl die Eigenschaften eines deterministischen wie auch die eines endlichen Automaten

Das Verhalten eines Automaten lässt sich gut mit einem Zustandsdiagramm beschreiben, dies ist ein gerichteter Graph, bestehend aus Knoten (Zustände) und gerichteten Kanten (Zustandsübergänge).
Um einen Zustand zu wechseln muss die Übergangsbedingung erfüllt sein.



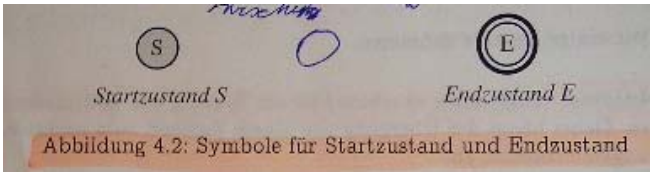
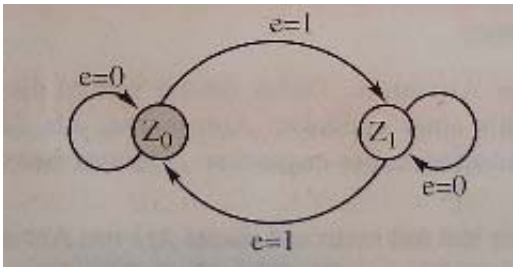
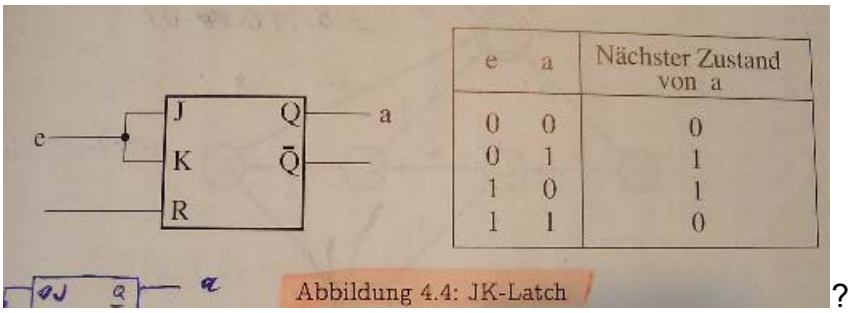


Abbildung 4.2: Symbole für Startzustand und Endzustand



Dieser Automat kann mit einem JK-Latch realisiert werden:



e	a	Nächster Zustand von a
0	0	0
0	1	1
1	0	1
1	1	0

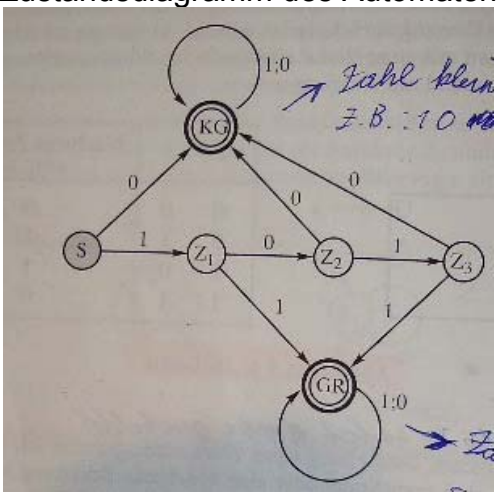
Abbildung 4.4: JK-Latch ?

Als nächstes wird ein Automat entworfen, der erkennt ob eine vier Bit lange Binärzahl kleiner oder gleich 10 ist.

dezimal	msb			lsb
⋮	⋮	⋮	⋮	⋮
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

müssen als > 10 erkannt werd

Zustandsdiagramm des Automaten:



Der Automat hat 6 Zustände, diese können mit drei D-Latches ($2^3 = 8$) dargestellt werden. Die Zustände sind in folgender Tabelle aufgelistet:

	Beschreibung	*Bedeutung*	D_2	D_1	D_0
S	Startzustand		0	0	0
Z_1	Zwischenzustand 1		0	0	1
Z_2	Zwischenzustand 2		0	1	0
Z_3	Zwischenzustand 3		0	1	1
KG	Endzustand	*kleiner gleich 10*	1	0	0
GR	Endzustand	*größer 10*	1	0	1

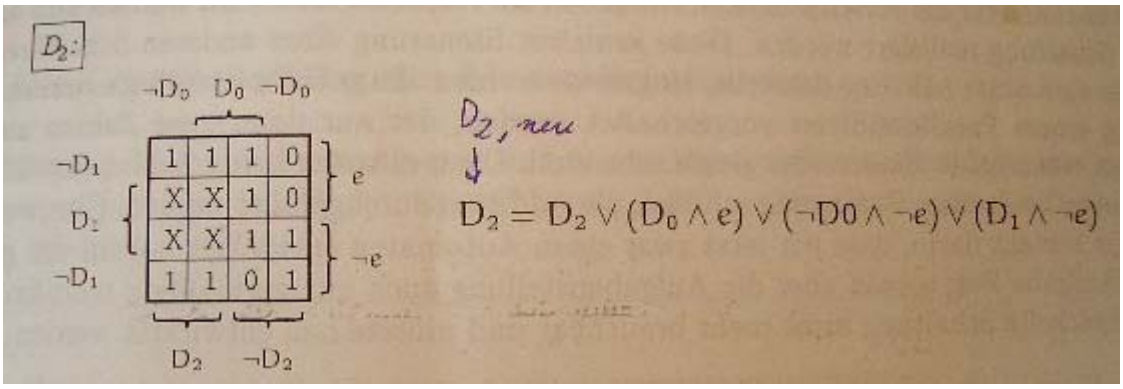
In der Tabelle wird jeweils der alte Zustand zusammen mit der Eingangsgröße und dem neuen Zustand als Folgezustand aufgelistet.

alter Zustand			Bedingung	neuer Zustand				
D_2	D_1	D_0	e	D_2	D_1	D_0	Zustandsübergang	Bedingung
0	0	0	0	1	0	0	S → KG	e = 0
0	0	0	1	0	0	1	S → Z_1	e = 1
0	0	1	0	0	1	0	Z_1 → Z_2	e = 0
0	0	1	1	1	0	1	Z_1 → GR	e = 1
0	1	0	0	1	0	0	Z_2 → KG	e = 0
0	1	0	1	0	1	1	Z_2 → Z_3	e = 1
0	1	1	0	1	0	0	Z_3 → KG	e = 0
0	1	1	1	1	0	1	Z_3 → GR	e = 1
1	0	0	0	1	0	0	KG → KG	e = 0
1	0	0	1	1	0	0	KG → KG	e = 1
1	0	1	0	1	0	1	GR → GR	e = 0
1	0	1	1	1	0	1	GR → GR	e = 1

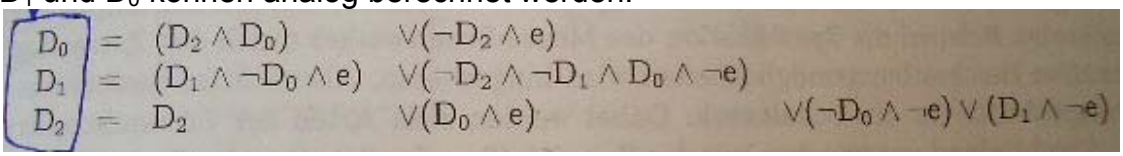
Daraus kann nun eine Schaltung entworfen werden, dazu stellt man für jedes Latch eine Funktion mit folgender Form auf:

$$D_i = f(D_2, D_1, D_0, e) \quad \text{mit } i = 0, 1, 2$$

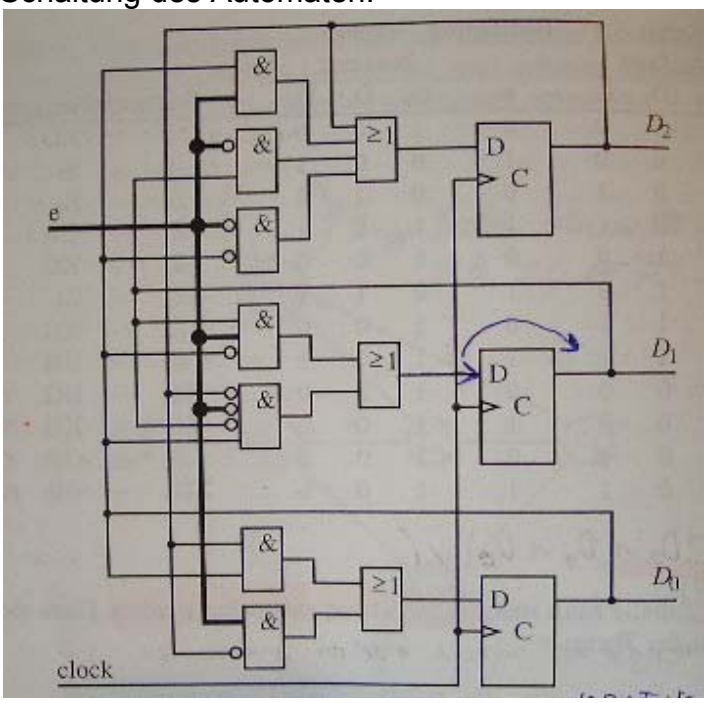
Der Wert in der Spalte „neuer Zustand“ von D_2 wird als Funktion der Spalten „alter Zustand“ von D_2, D_1, D_0 und e nach der disjunktiven Normalform in ein KV-Diagramm eingetragen:



D_1 und D_0 können analog berechnet werden:



Schaltung des Automaten:



<u>Das Moore-Schaltwerk</u>

Für viele Aufgabenstellungen werden sequenzielle Schaltwerke benötigt.

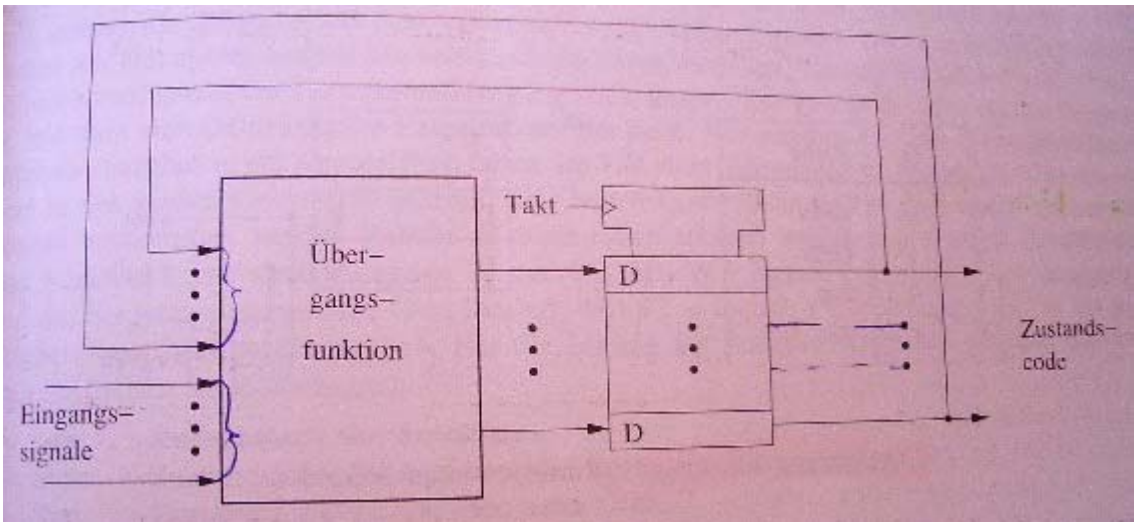
<u>Schaltwerk</u>

Unter einem Schaltnetz versteht man logische Funktionen ohne Speicherwirkung: Die Ausgänge eines Schaltnetzes hängen immer nur von den momentan anliegenden Eingangssignalen ab.

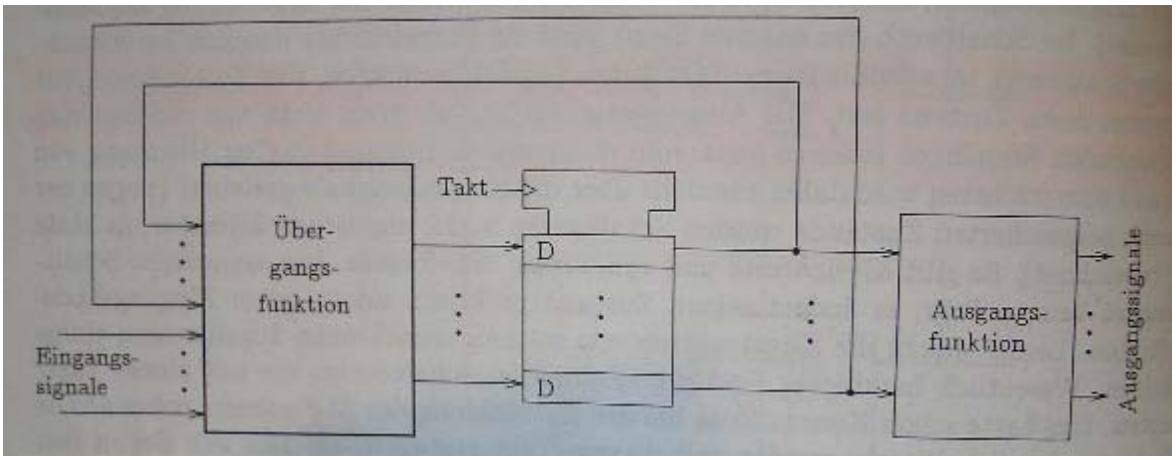
Ein Schaltwerk kann dagegen in verschiedenen Zuständen sein. Ein Zustandsspeicher hält den momentanen Zustand fest. Die Ausgangssignale hängen dann nicht nur von den momentan anliegenden Eingängen sondern auch vom

momentanen Zustand ab. Der Übergang von einem Zustand zum nächsten wird dabei ebenfalls über die Eingangssignale gesteuert.

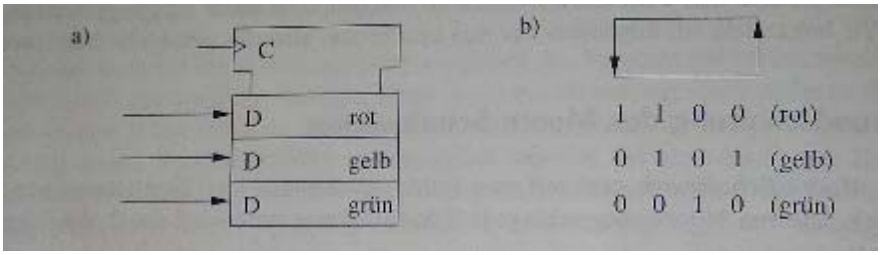
Die Grundschaltung des Moore-Schaltwerkes



Zur Speicherung des Zustandes werden D-Latches verwendet. Dieser Zustand bestimmt zusammen mit den anliegenden Eingangssignalen über die Übergangsfunktion den Folgezustand, der mit der nächsten schaltenden Taktflanke eingenommen wird. Ein Zustand kann mehrere Folgezustände haben.



Die Ausgangsfunktion bildet aus den Latch-Ausgängen den eigentlichen Ausgang. Mit Hilfe der Ausgangsfunktion lassen sich Latches einsparen.



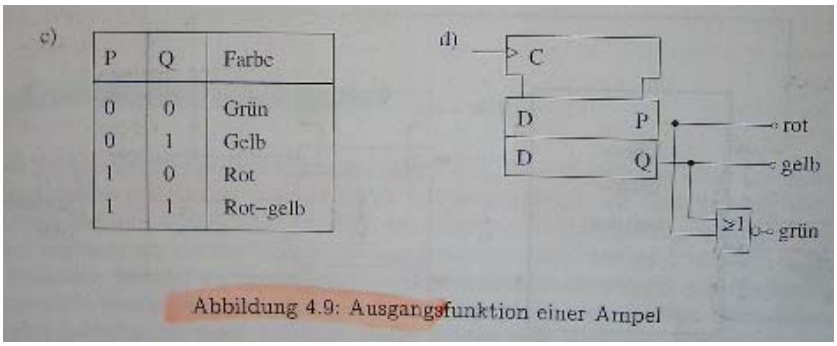


Abbildung 4.9: Ausgangsfunktion einer Ampel

Schaltwerksbeschreibung durch den Zustandsgraphen

Bei der Beschreibung eines Schaltwerks durch den Zustandsgraphen gehen wir von binären Ein- und Ausgangssignalen aus. Jedem Zustand muss ein Bitmuster der Latch-Ausgänge entsprechen.
Das soll an einem Beispiel eines Münzautomaten verdeutlicht werden:

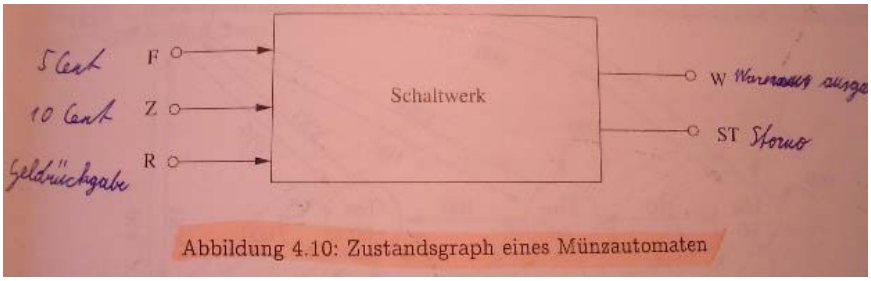


Abbildung 4.10: Zustandsgraph eines Münzautomaten

Das Schaltwerk hat drei Eingänge: F(fünf Cent), Z(zehn Cent) und R(Geldrückgabe), sowie zwei Ausgänge: W(Ware ausgeben) und ST(Storno).
Das Schaltwerk ist synchron.

Es werden fünf Zustände verwendet:
Idle: Ruhezustand
Five: 5 Cent wurden eingeworfen
Ten: 10 Cent wurden eingeworfen
Paid: vollständig Bezahlt
Cancel: Geldrückgabe

Zustandsgraph des Schaltwerks:

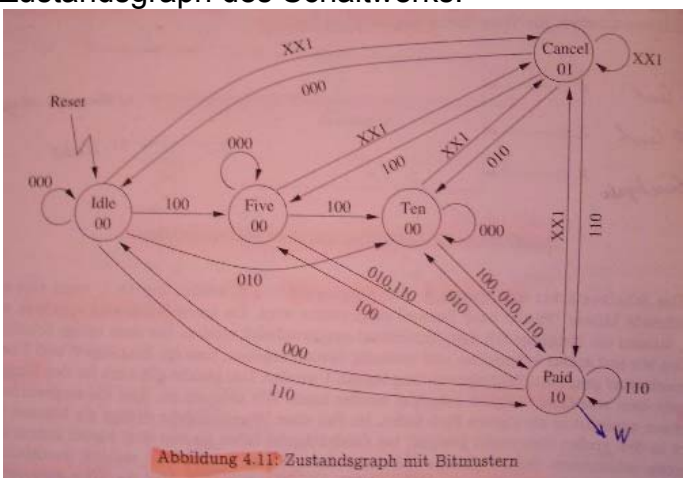


Abbildung 4.11: Zustandsgraph mit Bitmustern

Wird z.B. eine 5 Cent Münze eingeworfen geht das Schaltwerk in den Zustand Five über, werden z.B. ein Fünf und eine Zehn Cent Münze gleichzeitig eingeworfen, geht das Schaltwerk sofort in den Zustand Paid über und gibt die Ware aus.
Als Alternative kann der Zustandsgraph auch mit logischen Ausdrücken dargestellt werden:

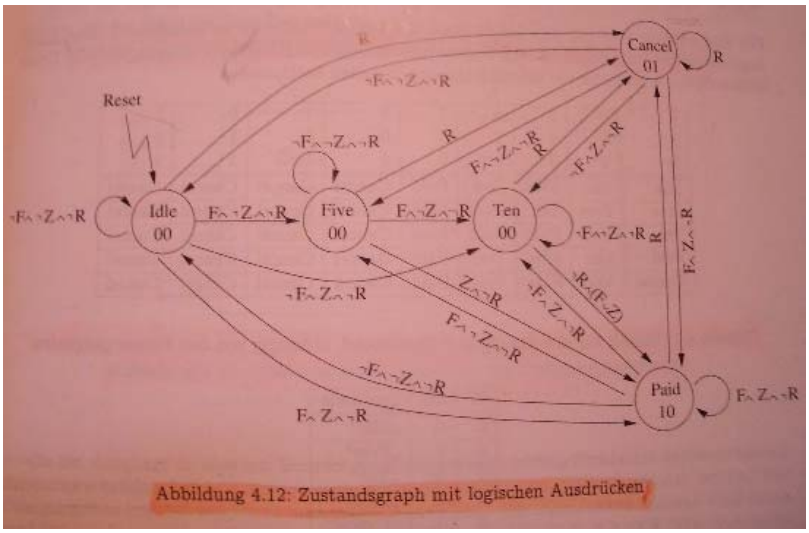


Abbildung 4.12: Zustandsgraph mit logischen Ausdrücken

Jeder Zustand des Schaltwerks entspricht einem Knoten des Zustandsgraphen, jeder Übergang zwischen zwei Zuständen einer Kante des Zustandsgraphen. Von jedem Zustand müssen Kanten für alle möglichen Eingangssituationen wegführen, ein und dieselbe Eingangssituation darf dabei nur an einer Kante vorkommen.

Alternativen zum Zustandsgraphen

Eine andere Möglichkeit ein Schaltwerk zu spezifizieren, ist eine Zustandsübergangstabelle und eine Ausgangstabelle.

F	0	1	0	1	0	1	0	1
Z	0	0	1	1	0	0	1	1
R	0	0	0	0	1	1	1	1
Idle	Idle	Five	Ten	Paid	Cancel	Cancel	Cancel	Cancel
Five	Five	Ten	Paid	Paid	Cancel	Cancel	Cancel	Cancel
Ten	Ten	Paid	Paid	Paid	Cancel	Cancel	Cancel	Cancel
Paid	Idle	Five	Ten	Paid	Cancel	Cancel	Cancel	Cancel
Cancel	Idle	Five	Ten	Paid	Cancel	Cancel	Cancel	Cancel

Tabelle 4.1: Tabelle für den jeweilgen Folgezustand, abhängig von den Eingangssignalen

W	ST	Zustände
0	0	„Idle“
0	0	„Five“
0	0	„Ten“
1	0	„Paid“
0	1	„Cancel“

Tabelle 4.2: Tabelle für die Ausgänge, abhängig nur vom Zustand

Realisierung mit „(1 aus n)“ und „dichter“ Zustandscodierung

Vor der Realisierung des Schaltwerks muss auf jeden Fall die Zustandskodierung festgelegt werden. Jedem Zustand muss ein Bitmuster der Latch-Ausgänge zugeordnet werden.

(1 aus n)-Zustandscodierung: für n Zustände werden n Latches verwendet
dichte Zustandscodierung: für n Zustände werden f (2^f <= n) Latchs verwendet

(1 aus n) Codierung:

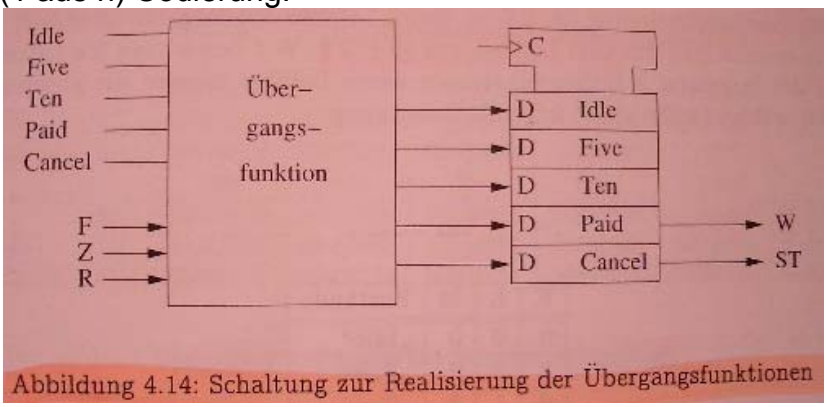
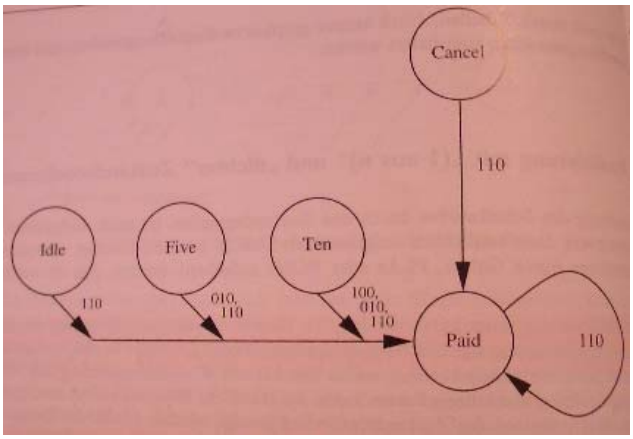


Abbildung 4.14: Schaltung zur Realisierung der Übergangsfunktionen

In diesem Beispiel lässt sich ein logischer Ausdruck für die Übergangsfunktion einfach und direkt aus dem Zustandsgraphen ablesen, deshalb brauch man hier keine Wahrheitstabelle.

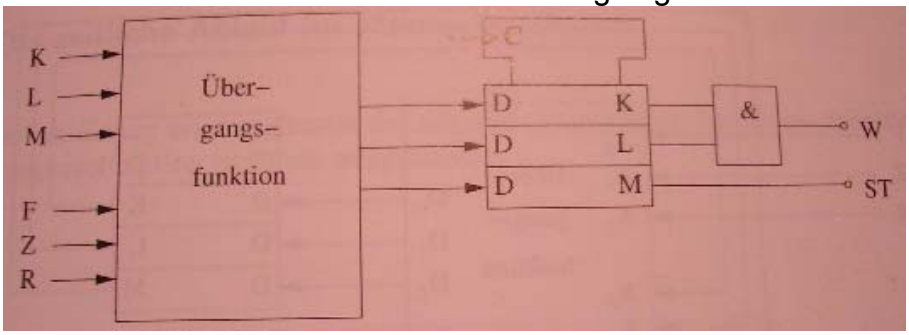


Dpaid=(idle^F^Z^¬R)v(F^Z^¬R)v(Z^(FvZ) ^¬R)v(cancel^F^Z^¬R)v(paid^F^Z^¬R)

dichte Zustandscodierung: drei Latches ($5<=2^3$)

K	L	M	Zustände
0	0	0	„Idle“
1	0	0	„Five“
0	1	0	„Ten“
1	1	0	„Paid“
0	0	1	„Cancel“

Das Schaltwerk sieht bei dichter Codierung folgendermaßen aus:



In der Zustandsübergangstabelle kann man für jeden Zustand den Folgezustand ablesen:

Eingänge	F	01010101	01010101	01010101	01010101	01010101	....
	Z	00110011	00110011	00110011	00110011	00110011	....
	R	00001111	00001111	00001111	00001111	00001111	....
alter Zustand	K	00000000	11111111	00000000	11111111	00000000	....
	L	00000000	00000000	11111111	11111111	00000000	....
	M	00000000	00000000	00000000	00000000	11111111	....
		Idle	Five	Ten	Paid	Cancel	
neuer Zustand	D_K	01010000	10110000	01110000	01010000	01010000	X X X
	D_L	00110000	01110000	11110000	00110000	00110000	X X X
	D_M	00001111	00001111	00001111	00001111	00001111	X X X

Tabelle 4.4: Tabelle der Zustandsübergänge

Für die Realisierung wird ein 5x64 ROM verwendet:

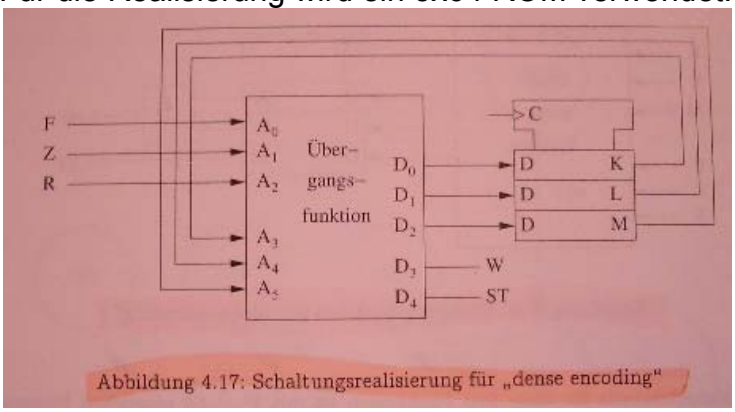


Abbildung 4.17: Schaltungsrealisierung für „dense encoding"

Der ROM Inhalt wird durch die Wahrheitstabelle der Übergangsfunktion bestimmt, nur die Ausgänge W und ST müssen noch ergänzt werden:

Adresse	0	8	16	24	32	≥ 40
A_0(F)	01010101	01010101	01010101	01010101	01010101	....
A_1(Z)	00110011	00110011	00110011	00110011	00110011	....
A_2(R)	00001111	00001111	00001111	00001111	00001111	....
A_3(K)	00000000	11111111	00000000	11111111	00000000	....
A_4(L)	00000000	00000000	11111111	11111111	00000000	....
A_5(M)	00000000	00000000	00000000	00000000	11111111	....
	Idle	Five	Ten	Paid	Cancel	
$D_0(D_K)$	01010000	10110000	01110000	01010000	01010000	0
$D_1(D_L)$	00110000	01110000	11110000	00110000	00110000	0
$D_2(D_M)$	00001111	00001111	00001111	00001111	00001111	0
D_3(W)	00000000	00000000	00000000	11111111	00000000	0
D_4(St)	00000000	00000000	00000000	00000000	11111111	0

Tabelle 4.5: Tabelle der Zustandsübergänge

Zusammenfassen kann gesagt werden: Eine (1 aus n)-Codierung bracht mehr Latches als die dichte Codierung, in machen Fällen ergeben sich bei der (1 aus n)-Codierung jedoch sehr einfache Funktionen.

Der Zeitliche Ablauf im Moore-Schaltwerk

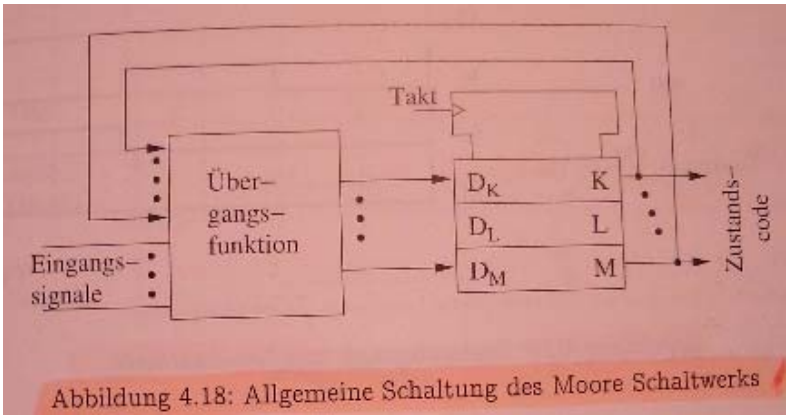
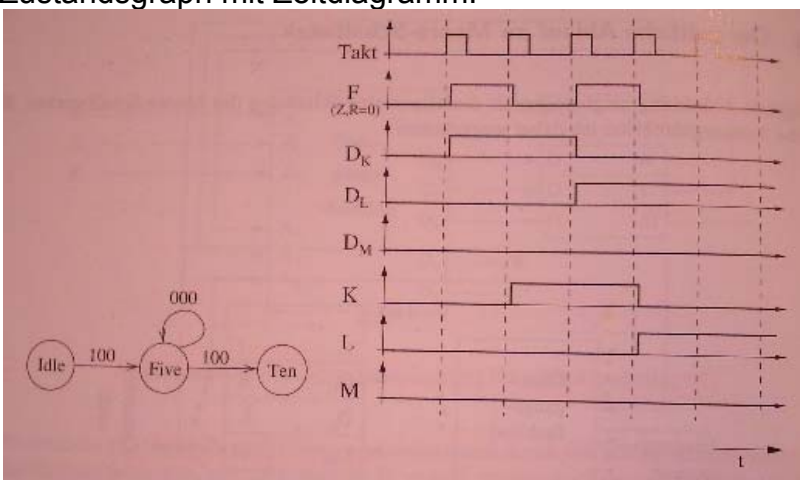


Abbildung 4.18: Allgemeine Schaltung des Moore Schaltwerks

Betrachten wir einen sehr ungünstigen Fall: Die D-Eingänge ändern sich gerade zum Zeitpunkt der Taktflanke. Dann kann nicht vorausgesagt werden, welche Werte die Latches annehmen; dadurch könnte ein falsches Bitmuster entstehen, das in der Zustandscodierung nicht vorgesehen ist. Für die korrekte Funktion des Schaltwerkes müssen die neuen Eingänge offensichtlich schon um die Durchlaufzeit plus der Latch-Vorbereitungszeit vor der schaltenden Taktflanke im eingeschwungenen Zustand vorliegen.

Zustandsgraph mit Zeitdiagramm:



Da die Eingänge den Zustand nicht sofort ändern, können sie auch nicht direkt auf die Ausgänge einwirken, da diese ja nur vom Zustand abhängen. Die maximale Taktfrequenz eines Schaltwerkes lässt sich wie folgt berechnen:

$T_{Min} = t_{Latch}$ (Durchlaufzeit der Latches) + t_{Gate} (Durchlaufzeit durch die Übergangsfunktion) + t_{setup} (Latch Vorbereitungszeit)

Da alle Latches gleichzeitig einschalten geht diese Zeit nur einmal in die Rechnung ein.

Maximale Taktfrequenz: $f_{max} = 1/T_{min}$

Am sichersten funktioniert das Schaltwerk, wenn sich die Eingänge nur zum Zeitpunkt der schaltenden Taktflanke ändern: Es steht dann das ganze Taktintervall zum Bilden der neuen Vorbereitungen zur Verfügung.

Synchronisierung von asynchronen Eingangssignalen

Signale die von aussen kommen, sind in der Regel asynchron zum Takt. Wir müssen also asynchrone Signale mit einem Takt synchronisieren. Die einfachste Möglichkeit dazu ist, das asynchrone Eingangssignal über ein zusätzliches D-Latch zu führen.

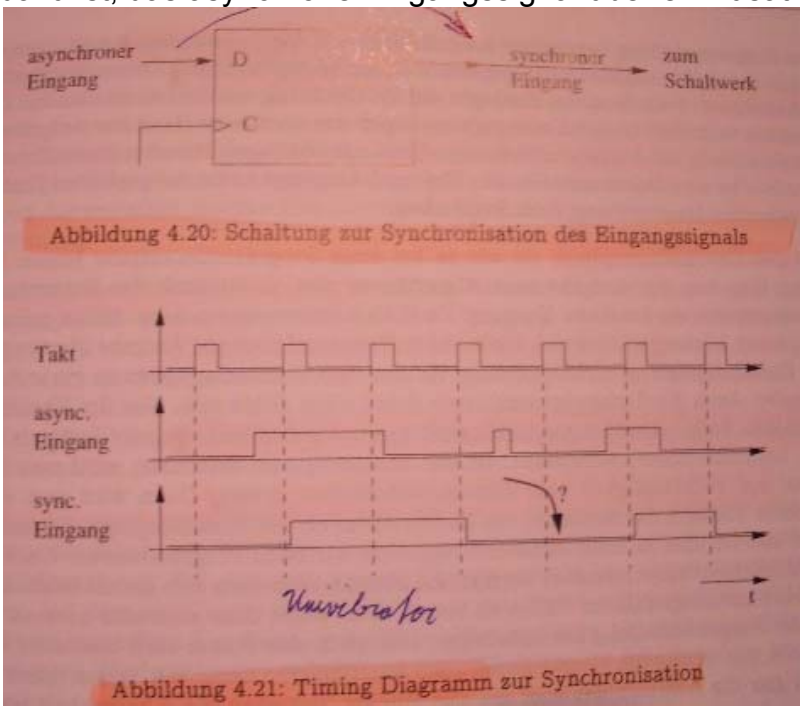


Abbildung 4.20: Schaltung zur Synchronisation des Eingangssignals

Abbildung 4.21: Timing Diagramm zur Synchronisation

Ein asynchrones Signal kann auch verloren gehen. Um dies zu vermeiden, kann man sogenannte „Impulsformerstufen“ einsetzen, die das „zu kurze“ Signal um mehr als eine Taktperiode verlängern.

Systematische Schaltwerksentwicklung

Aufbereiten der Aufgabenstellung: festlegen der Ein- und Ausgangssignale und der Zustande des Schaltwerks
Entwurf des Zustandsgraphen: erstellung des Zustandsgraphen, prüfung auf Vollständigkeit und Widerspruchsfreiheit, Mindestanzahl von Zuständen
Minimierung der Zustandsanzahl: nur bei sehr großen Schaltwerken
Festlegen der Zustandscodierung: festlegung der Anzahl der Latches und die Codierung der einzelnen Zustände, 1 aus n oder dichte Codierung, Tabelle der Zustandscodierung
Übergangs- und Ausgabefunktion: entscheidung über die Realisierung der Funktion (Gattern, PLAs oder ROM), festlegung der endgültigen Funktionen
Dokumentation der Gesamtschaltung
Berechnung der Maximalen Taktfrequenz

Beispiel: 3-Bit-Zähler mit JK-Latches

Es soll ein Dualzähler mit drei Ausgängen erstellt werden, die Ausgänge sollen im Zyklus von 000 bis 111 um eine Zahl weiterschalten. Steuernde Eingänge gibt es keine. Die Speicherung soll mit JK-Latches erfolgen, die Übergangsfunktion ist mit Gattern aufzubauen.
Es gibt drei Ausgänge: A, B, C und es sind 8 Zustände (0 … 7) notwendig.

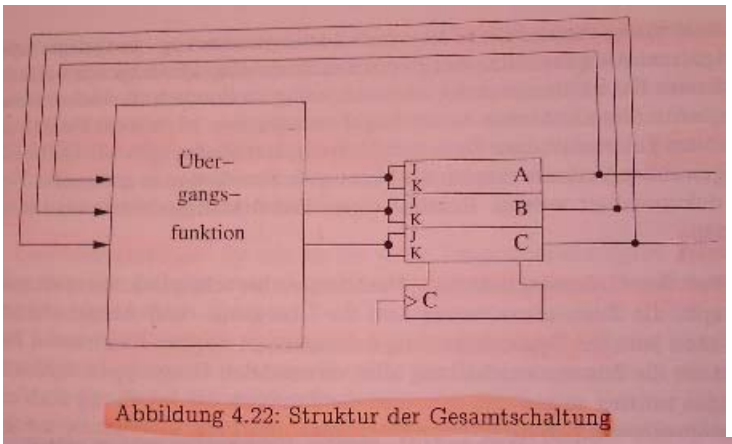


Abbildung 4.22: Struktur der Gesamtschaltung

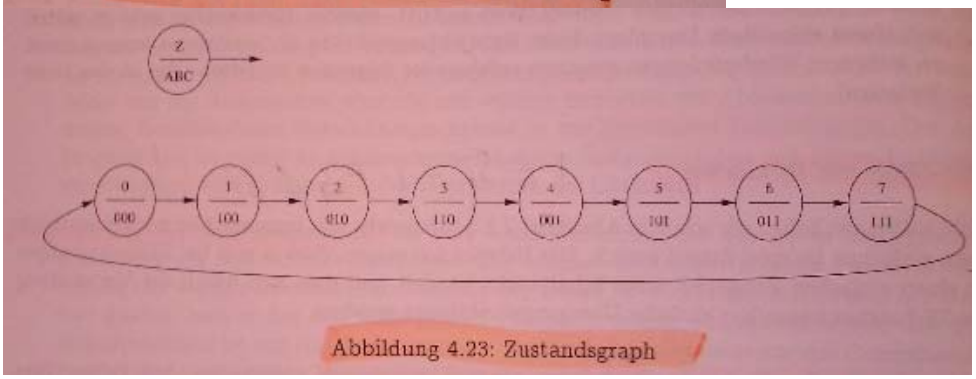


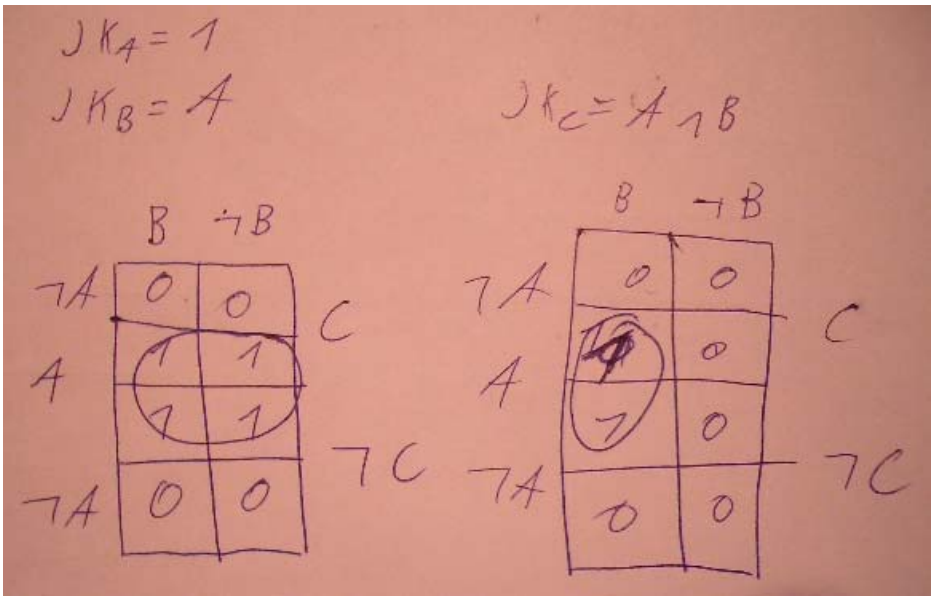
Abbildung 4.23: Zustandsgraph

Es wird ein dichte Zustandscodierung gewählt:

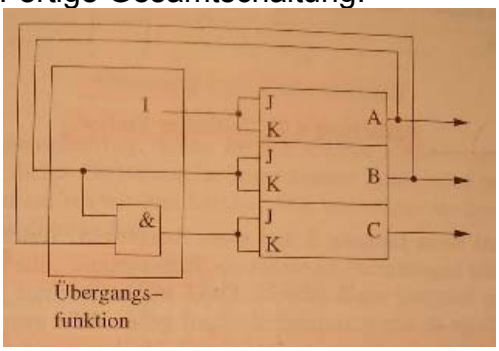
	A	B	C
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
⋮		⋮	
7	1	1	1

Aus der Übergangsfunktion ergeben sich folgende Formeln:

alter	A	0	1	0	1	0	1	0	1
Zustand	B	0	0	1	1	0	0	1	1
	C	0	0	0	0	1	1	1	1
gewünschter	A	1	0	1	0	1	0	1	0
neuer	B	0	1	1	0	0	1	1	0
Zustand	C	0	0	0	0	1	1	1	0
dazu	JK_A	1	1	1	1	1	1	1	1
nötige	JK_B	0	1	0	1	0	1	0	1
Vorbereitung	JK_C	0	0	0	1	0	0	0	1



Fertige Gesamtschaltung:



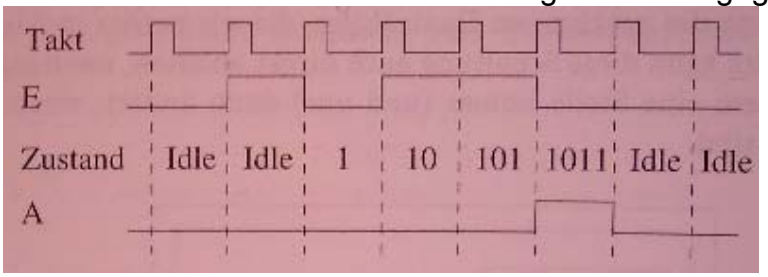
Berechnung der maximalen Taktfrequenz:

Durchlaufzeit der Latches:	45	ns
Durchlaufzeit der Übergangsfunktion:	15	ns
Vorbereitungszeit der Latches:	5	ns
Summe	65	ns

$1/(65*10^{-6}) = 15{,}4\text{Mhz}$

<u>Beispiel: Erkennung einer Eingangsfolge</u>

Ein Schaltwerk hat einen Eingang E und einen Ausgang A. Der Ausgang A soll mit einer 1 anzeigen, dass am Eingang die Eingangsfolge 1011 aufgetreten ist. Bei einer Eingangsfolge 10111011 soll am Ausgang 2 mal eine 1 anliegen, bei einer Überlappung (1011011) soll nur einmal eine 1 anliegen.
Da bei einem Moore Schaltwerk die Ausgänge nicht sofort auf die Eingänge reagieren können, kann die 1 am Ausgang frühestens in dem Taktintervall nach dem letzten Einser der zu erkennenden Folge 1011 ausgegeben werden.



Man braucht für die Lösung folgenden Zustände:
Idle (Ruhezustand) und die Zustände: 1, 10, 101, 1011
Insgesamt also fünf Zustände.

Bei nur einem Eingang müssen von jedem Knoten zwei Kanten wegführen, eine für Eingang 0, die andere für Eingang 1.

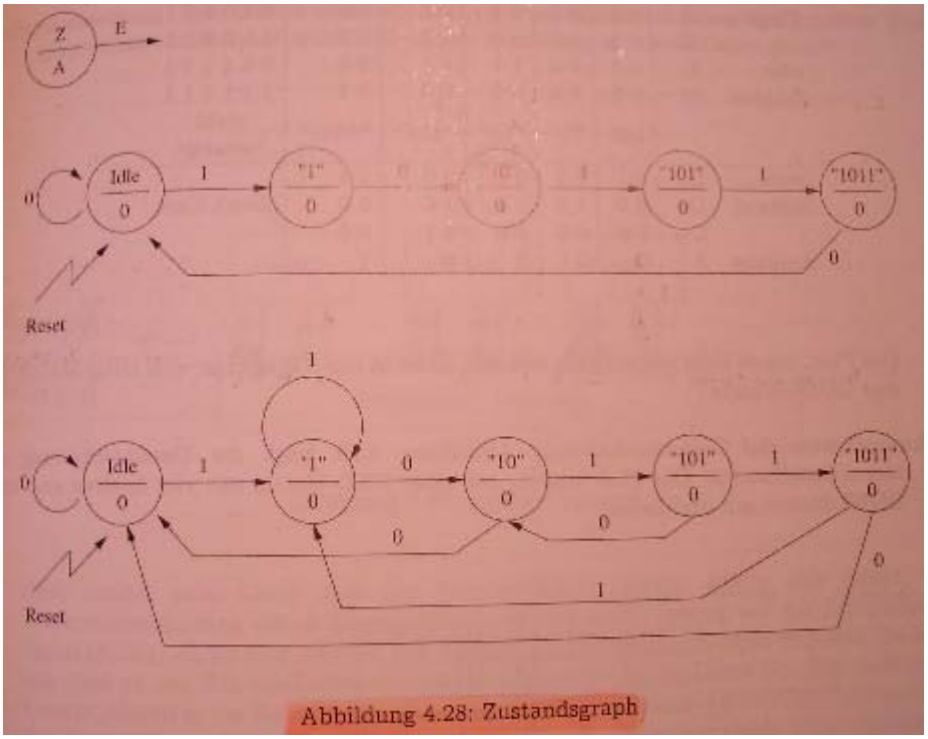


Abbildung 4.28: Zustandsgraph

Dichte Zustandscodierung:

K	L	M	Zustände
0	0	0	„Idle“
1	0	0	„1“
0	1	0	„10“
1	1	0	„101“
0	0	1	„1011“

Eine Ausgangsfunktion ist nicht notwendig, weil der Ausgang des M-Latches direkt als Ausgang genommen werden kann.

Eingang	E	0 1	0 1	0 1	0 1	0 1	0 1 0 1 0 1
	K	0 0	1 1	0 0	1 1	0 0	1 1 0 0 1 1
alter	L	0 0	0 0	1 1	1 1	0 0	0 0 1 1 1 1
Zustand	M	0 0	0 0	0 0	0 0	1 1	1 1 1 1 1 1
		Idle	"1"	"10"	"101"	"1011"	nicht benutzt!
neuer	D_K	0 1	0 1	0 1	0 0	0 1	
Zustand	D_L	0 0	1 0	0 1	1 0	0 0	Don't Care
	D_M	0 0	0 0	0 0	0 1	0 0	
Ausgang	A	0	0	0	0	1	

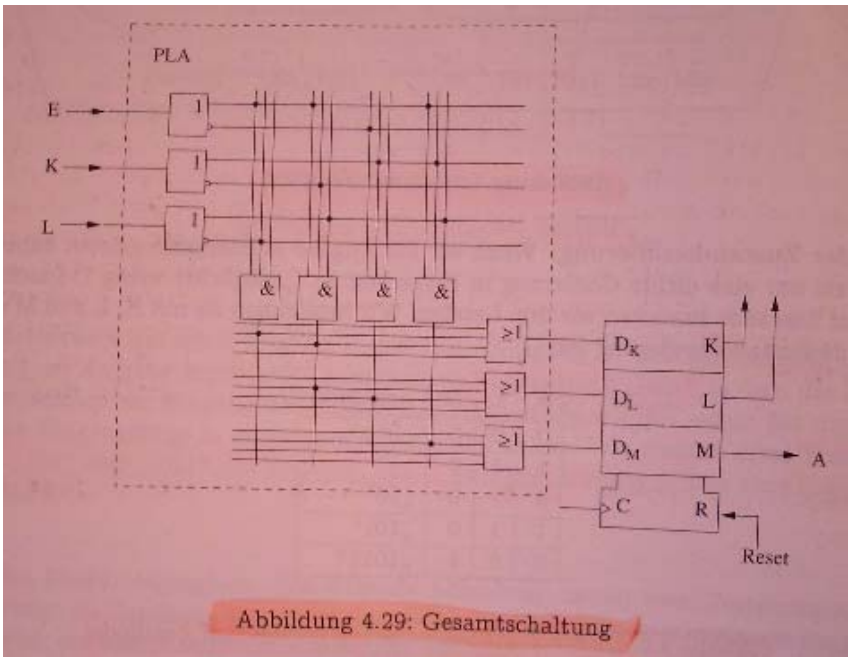
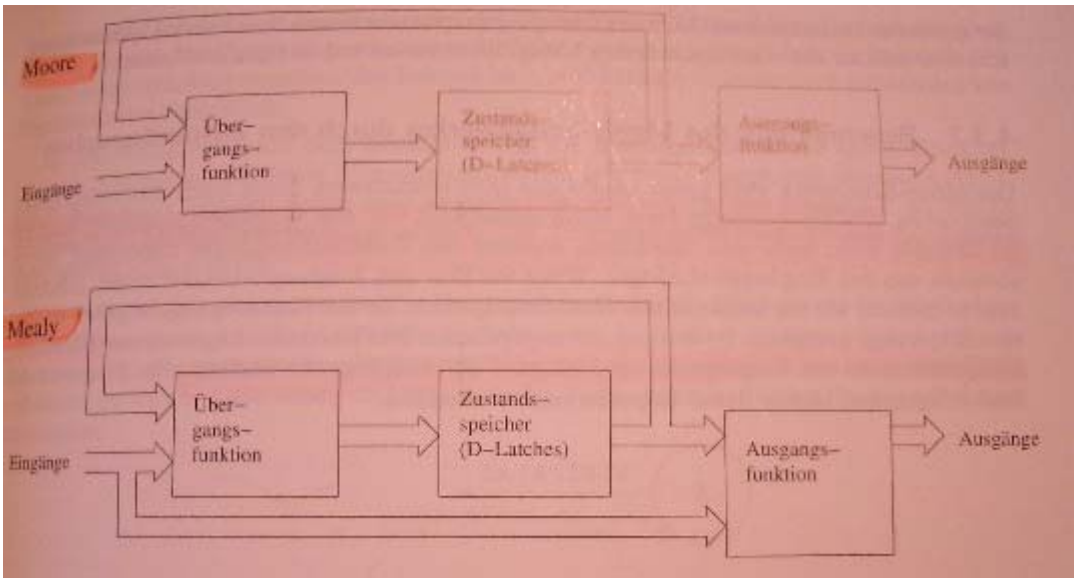


Abbildung 4.29: Gesamtschaltung

Das Mealy-Schaltwerk

Im Prinzip unterscheidet sich das Mealy Schaltwerk vom Moore Schaltwerk nur durch eine andere Ausgangsfunktion.

Die Schaltung eines Mealy Schaltwerks

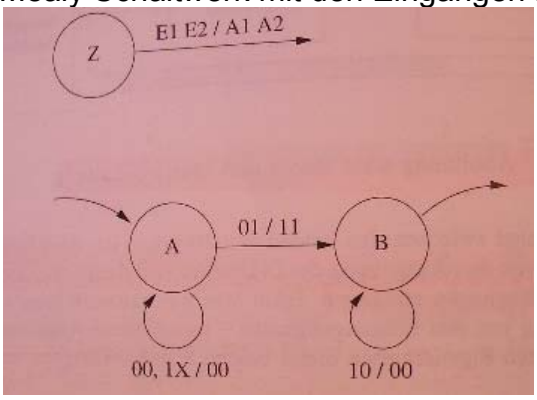


Beim Mealy Schaltwerk hängen die Ausgänge nicht nur von den Ausgängen der D-Latches und damit vom momentanen Zustand, sondern auch von den Eingängen ab. Ein Mealy Schaltwerk kann also sofort auf die Änderung der Eingänge an den Ausgängen reagieren. Typischerweise hat ein Mealy Schaltwerk weniger Zustande und es sind weniger Taktzyklen notwendig als bei einem Moore Schaltwerk.
Die Ausgangsfunktion ist beim Mealy Schaltwerk meistens wesentlich komplexer als beim Moore Schaltwerk.

Beschreibung des Mealy-Schaltwerks durch den Zustandgraphen

Da die Ausgänge beim Mealy-Schaltwerk von den Eingängen abhängen, sind sie im Graphen nicht mehr den Zuständen, sondern den Zustandsübergängen zugeordnet.

Mealy Schaltwerk mit den Eingängen E1 und E2 und den Ausgängen A1 und A2:



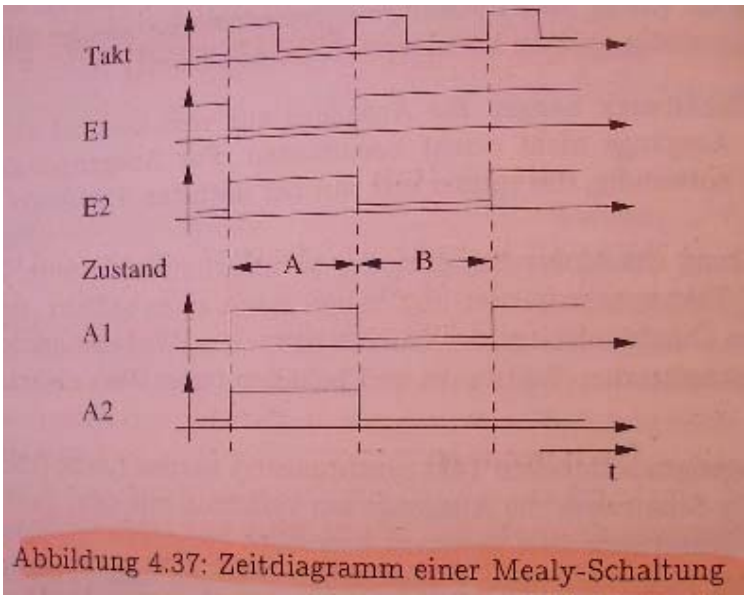


Abbildung 4.37: Zeitdiagramm einer Mealy-Schaltung

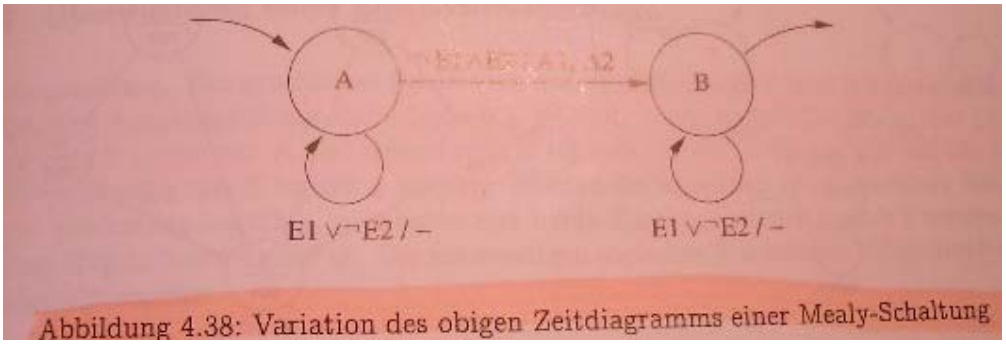
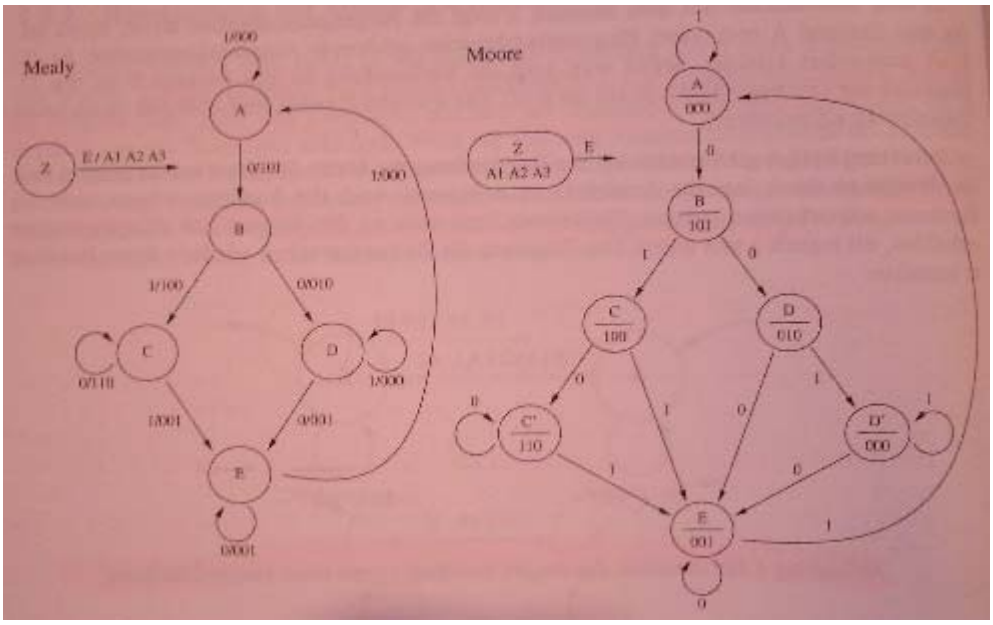


Abbildung 4.38: Variation des obigen Zeitdiagramms einer Mealy-Schaltung

Mealy-Moore-Transformation

Ein Moore Schaltwerk ist ein Spezialfall eines Mealy Schaltwerks. Die Umwandlung eines Mealy in ein Moors Schaltwerk ist jedoch nicht trivial und gar nicht 1:1 möglich. Eine Transformation soll an einem Beispiel gezeigt werden:



Vom Zustand A aus sind bei der Mealy Variante die Ausgangsbitmuster 000 und 101 möglich. Bei 000 verbleiben wir in A. Wir weisen deshalb dieses Ausgangsbitmuster bei der Moore-Variante dem Zustand A zu. Das Bitmuster 101 verlegen wir in den Zustand B. Hier laufen die Schaltwerke bereits zeitlich auseinander.
Vom Zustand B aus können beim Mealy Schaltwerk die Ausgangsbitmuster 100 und 010 entstehen. Diese können wir in der Moore Variante den Zuständen C und D zuweisen. Allerdings verbleibt das Mealy Schaltwerk für Eingang 0 im Zustand C und gibt dabei 110 ab. Den Zustand C haben wir in der Moore Variante jedoch schon mit 100 belegt. Wir müssen daher für das Ausgangsbitmuster 110 einen Zusatzzustand C‘ einführen.
Die Transformation geht so weiter (siehe Buch Seite 120)

<u>Die Maximale Taktfrequenz des Mealy Schaltwerks</u>

Hier gibt es im Vergleich zum Moore Schaltwerk nicht neues.

<u>Beispiel: Überwachung der Einschaltreihenfolge</u>

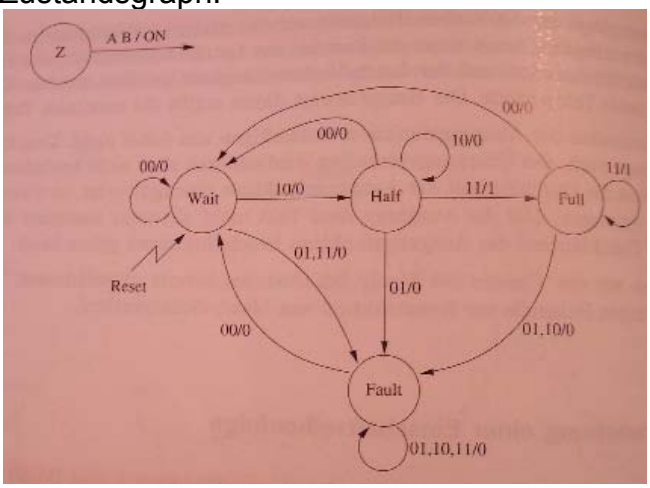
Ein synchrones Schaltwerk hat zwei Eingänge A und B und einen Ausgang ON. Der Ausgang soll nur dann logisch 1 werden, wenn vom Ruhezustand aus (A und B auf logisch 0) erst nur A und später auch B logisch 1 wird. In diesem Fall soll der Ausgang ON gleichzeitig mit B logisch 1 werden.

Da der ON Ausgang gleichzeitig mit dem B Eingang logisch 1 werden soll, ist jedenfalls ein Mealy Schaltwerk notwendig.

Es sind 4 Zustände zum lösen der Aufgabe notwendig:

Wait Ruhezustand
Half Signal A alleine wurde eingeschaltet
Full Auch Signal B wurde eingeschaltet
Fault Falsche Einschaltreihenfolge, Warten auf 00 beim Abschalten

Zustandsgraph:



Dichte Zustandscodierung, wir verwenden zwei Latches P und Q:

P	Q	Zustände
0	0	Wait
1	0	Half
0	1	Fault
1	1	Full

Zustandsübergangstabelle:

A	0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1
B	0 0 1 1	0 0 1 1	0 0 1 1	0 0 1 1
P	0 0 0 0	1 1 1 1	0 0 0 0	1 1 1 1
Q	0 0 0 0	0 0 0 0	1 1 1 1	1 1 1 1
	Wait	Half	Full	Fault
D_P	0 1 1 1	0 1 1 0	0 1 1 0	0 1 1 1
D_Q	0 0 1 1	0 0 1 1	0 1 1 1	0 1 1 1
ON	0 0 0 0	0 0 0 1	0 0 0 1	0 0 0 0

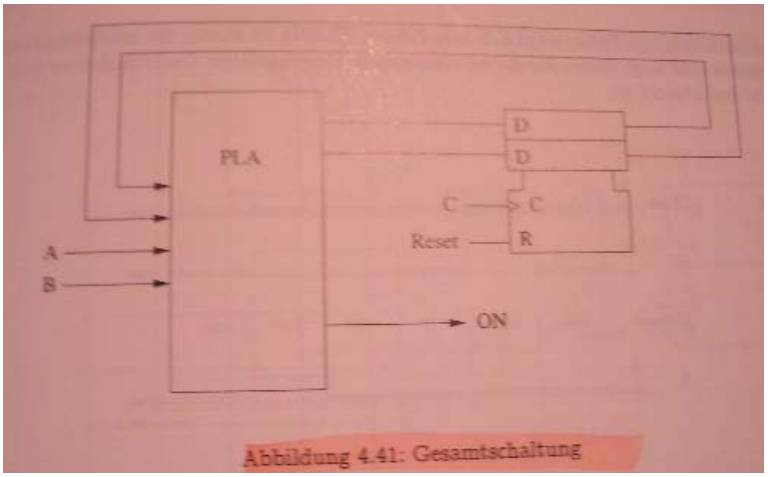
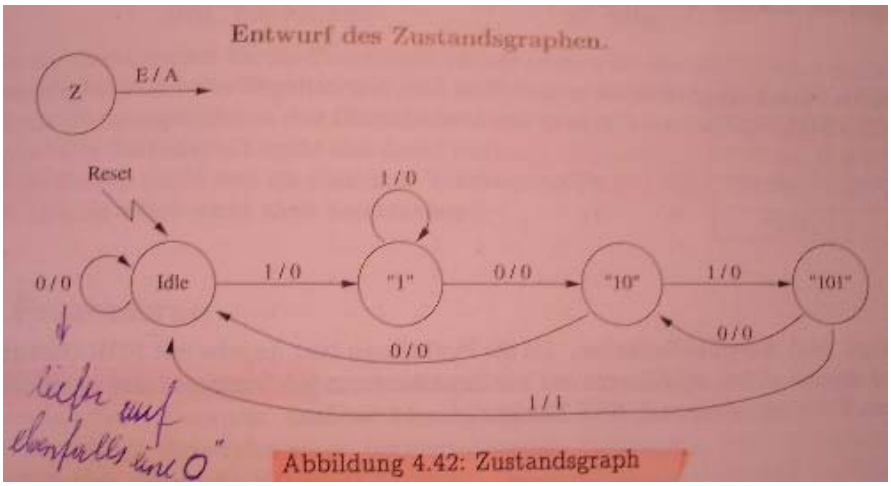


Abbildung 4.41: Gesamtschaltung

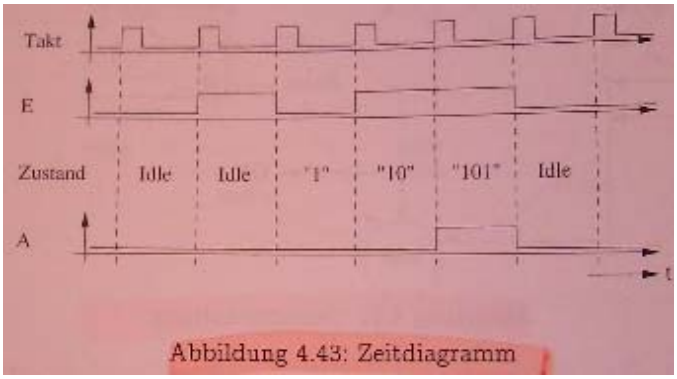
Beispiel: Erkennen der Eingangsfolge 1011

Im Gegensatz zum Moore Schaltwerk benötigen wir zur Lösung dieser Aufgabe mit dem Mealy Schaltwerk nur vier Zustände: Idle, 1, 10, 101



Ist im Zustand 101 der Eingang logisch 1, wird eine logisch 1 ausgegeben.

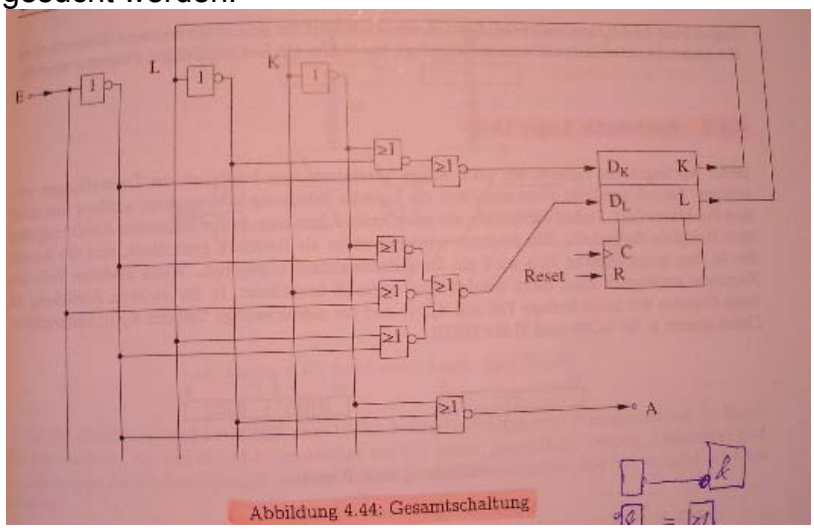
Zeitlicher Ablauf:



Zustandscodierung: (jetzt werden nur mehr 2 Latches benötigt)

K	L	Zustände
0	0	Idle
1	0	„1“
0	1	„10“
1	1	„101“

Es war gefordert die Funktionen in konjunktiver, also in ODER/UND und nicht wie sonst in UND/ODER Form darzustellen, dafür muss im KV Diagramm nach 0 Blöcken gesucht werden:



Die Maximale Taktfrequenz wird wie folgt berechnet:

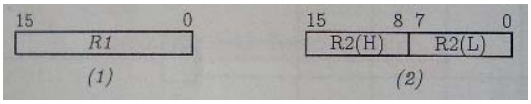
$t_{Latch} + t_{Gate} + t_{setup}$ = 40ns + (10ns (1x Negation) + 15ns + 15ns (2x NOR-Gatter)) + 10ns = 90ns

$1/(90 * 10^{-6})$ = 11,1 Mhz

Prozessoren

Arithmetic Logic Unit (ALU)

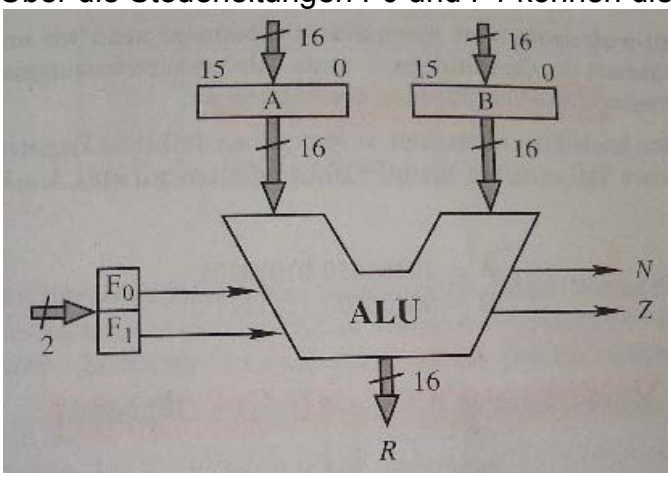
Es soll uns jetzt nicht mehr die logische Schaltung interessieren sondern nur mehr ihre Funktionalität. Ein Register wird nur mehr als Rechteck dargestellt.



Für die ALU wird ein eigenes graphische Sympol vorgestellt, die Funktionalität muss jedoch noch spezifiziert werden:

Parallele Addition
Bitweise UND-Verknüpfung
Bitweise Komplementbildung
Unverändertes durchschalten eines Datenwortes

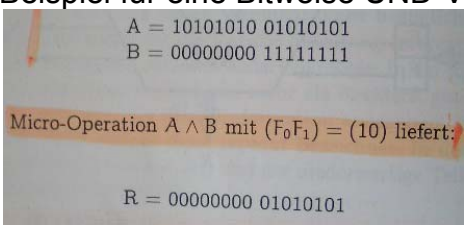
Über die Steuerleitungen F0 und F1 können die vier Funktionen angewählt werden.



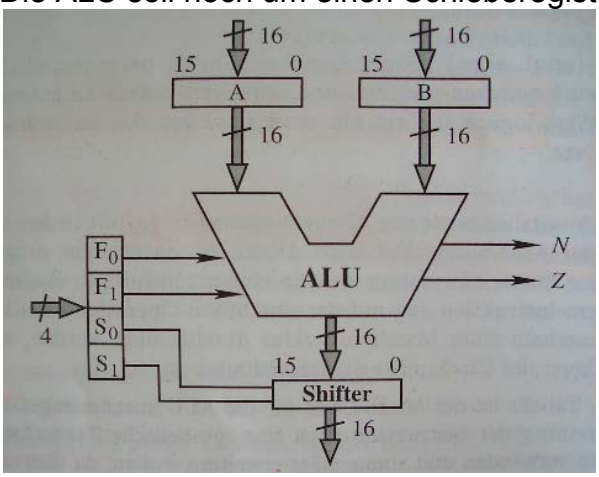
In den Registern A und B sind die Datenwörter gespeichert die die ALU verarbeiten soll, R ist der Ausgang für die Resultate.
Die Übertragsanzeige zeigt den Übertrag beim Rechnen an und die Nullanzeige wird logisch 1 wenn im Ergebnis alle Bits logisch 0 sind.
Die Vorzeichenanzeige underscheidet zwischen positiven und negativen Werten: logisch 0 → positiver Wert, logisch 1 → negativer Wert.

micro instruction (F_0F_1)	Beschreibung	symbolisch
(00)	A unverändert durchschalten	R←A
(01)	A und B addieren	R←A+B
(10)	A und B bitweise ∧-verknüpfen	R←A∧B
(11)	A negieren	R←¬A

Beispiel für eine Bitweise UND-Verknüpfung:



Die ALU soll noch um einen Schieberegister erweitert werden:



Um die drei neuen Kommandos: shift left, shift right oder keine Operation anzusprechen werden zusätzlich zwei Steuereingänge (S0 S1) hizugefügt:

(S_0S_1)	Beschreibung	symbolisch
(00)	keine Veränderung	SH←R
(01)	shift left	SH←lsh(R)
(10)	shift right	SH←rsh(R)
(11)	**nicht gültig**	–

Wegen der ungültigen Shifter Funktion (11) existieren nur zwölf statt der bei vier Bit möglichen 2^4 Instruktionen:

micro instruction $(F_0F_1S_0S_1)$	symbolisch
(0000)	SH← A
(0001)	SH← lsh(A)
(0010)	SH← rsh(A)
(0100)	SH← A+B
(0101)	SH← lsh(A+B)
(0110)	SH← rsh(A+B)

micro instruction $(F_0F_1S_0S_1)$	symbolisch
(1000)	SH← A ∧ B
(1001)	SH← lsh(A ∧ B)
(1010)	SH← rsh(A ∧ B)
(1100)	SH← ¬A
(1101)	SH← lsh(¬ A)
(1110)	SH← rsh(¬ A)

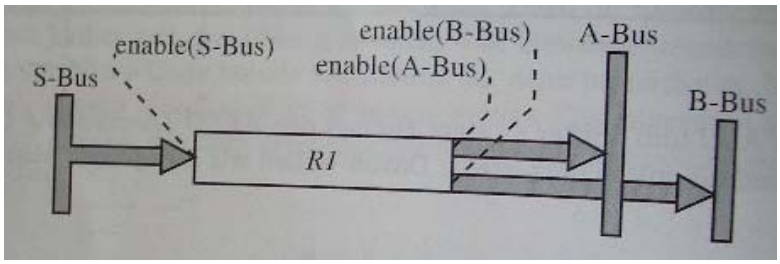
<u>Register File und Busverbindungen</u>

Nun wird ein Konzept entwickelt das nicht nur die Register A und B verarbeiten kann, sondern das mehrere gleichberechtigte Register vorsieht.
Es werden mehrere parallele Leitungen (Informationspfade), für jedes Bit eine Leitung, von jedem Register zu A und B gelegt. Die Datenpfade nennt man Busverbindung oder kurz Bus. Eine übergeordnete Logik entscheidet dann, welcher

Speicher die Verbindung benutzen darf, diese Steuerung nennt man allgemein Bus Arbitration Logic oder Bus Arbiter.
In unserem Fall kommen drei Busverbindungen zum Einsatz:

A- und B-Bus verbinden die Speicher mit dem A- bzw. B-Register.
Das Ergebnis wird über den S-Bus von der Shifter Einheit an alle Register übergeben werde

Unsere CPU hat 16 Register, mit Hilfe eines Decoders kann man die Information, welches Register seine Daten auf den Bus legen soll, mit Hilfe von 4 Bit pro Bus codieren:



13 der 16 Register können ohne Einschränkung verwendet werden, drei Register sind für die Konstanten 0, +1 und -1 reserviert und können nur gelesen werden, das Register mit der Adresse 15 dient als Accumlator. In den Registern A, B und S werden die Adressen binär eingetragen um die Operanden und das Ergebnisregister jeder Prozedur festlegen zu können.
Eine solche Konfiguration nennt man Scratchpad oder Register File.

Die Steuerleitungen C1, C2 und C4 sorgen mit Hilfe von zeitlich gestaffelten Impulsen für eine korrekte Befehlsausführung. Bei einer Operation der Art R1 <-- R1 + R2 könnte ohne steuerleitung ein Fehler passieren.
Die Control Unit enthält eine Schaltung, welche die geeigneten Impulse für die Steuerleitungen C1, C2 und C4 erzeugt.

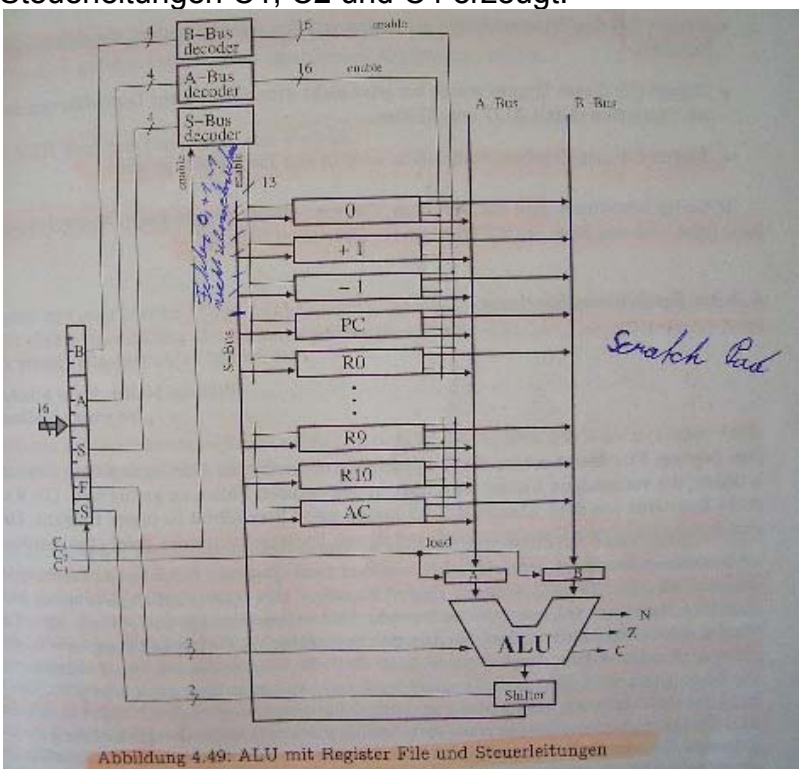


Abbildung 4.49: ALU mit Register File und Steuerleitungen

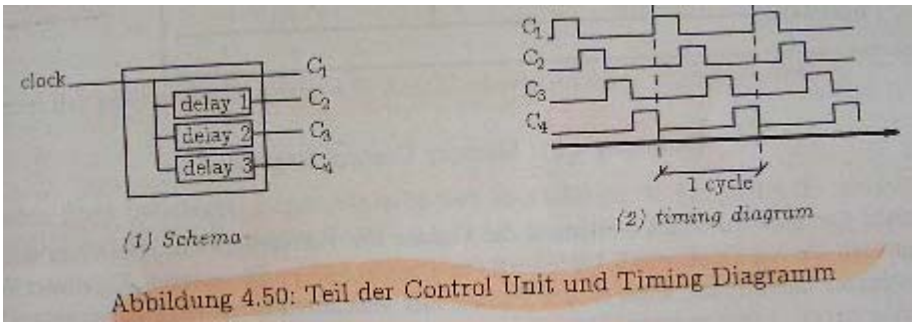


Abbildung 4.50: Teil der Control Unit und Timing Diagramm

Trigger C1: die aktuelle Mikro Instruktion wird geladen, Daten werden auf A- und B-Bus gelegt, es erfolgt die Auswahl der Funktion der ALU und des Shift-Registers.
Trigger C2: die Register A und B werden mit den Daten versorgt, die sich auf den Bussen befinden
Trigger C4: das Ergebnis vom S-Bus wird in das Zielregister geladen

Speicheranbindung

Die Speichermöglichkeit im Register-File ist für die meisten Fälle zu gering, deshalb soll nun eine Schnittstelle zum Datenaustausch mit Speicherbausteinen geschaffen werden.
Das Modell wird um zwei neue Register ergänzt: MAR(Memory Address Register) und MBR(Memory Buffer Register). Die Speicherbausteine sind mit dem Adress- und Datenbus verbunden. Am Adressbus kann stets die Information des MAR abgelesen werden. Zur Festlegung der Datenübertragungsrichtung und des Zeitpunktes, zu dem der Transfer stattfinden soll, sind die beiden Steuerleitungen read/write und memory select notwendig.
Im Control- oder Steuerbus sind alle Steuerleitungen zusammengefasst.

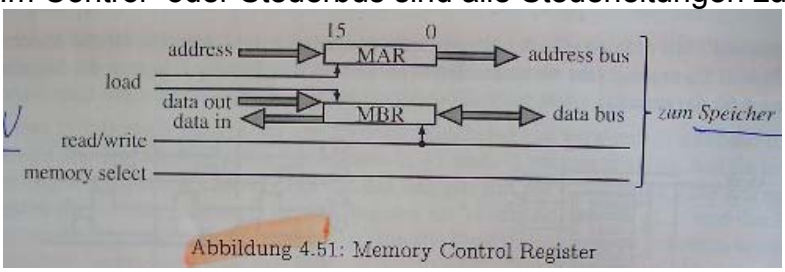


Abbildung 4.51: Memory Control Register

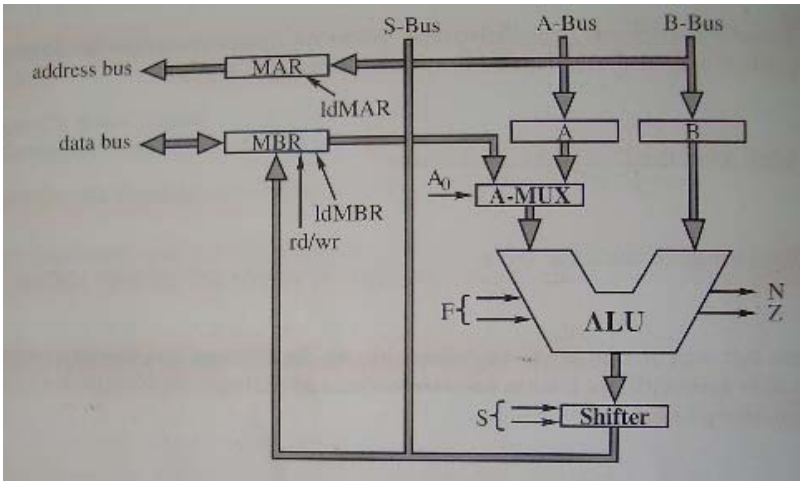
Bei einer Länge von 16 Bit lassen sich die Adressen von 0 bis $2^{16} - 1$ (=64K) adressieren.

Der Lesevorgang funktioniert wie folgt:
1. Die Adresse der gewünschten Speicherzelle wird in das MAR geschrieben, gleichzeitig wird der Kontrolleingang read und memory select aktiviert.
2. Die Zeitspanne welche die Speicherbausteine benötigen um die Information auf den Datenbus zu legen (Speicherzugriffszeit) muss abgewartet werden.
3. Das MBR kann das Datenwort nun einlesen.

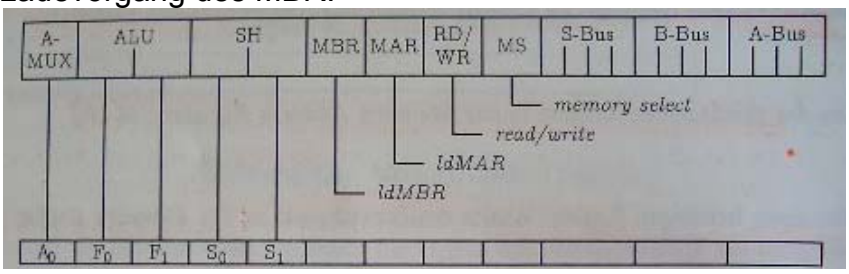
Der Schreibvorgang funktioniert wie folgt:
1. Im MAR wird die Adresse abgelegt wo die Information gespeichert werden soll, stellt aber zusätzlich das Datum im MBR bereit und aktiviert die Kontrollleitungen write und memory select.
2. Während der Speicherzugriffszeit müssen sowohl MAR als auch MBR die richtigen Werte enthalten, damit die Daten richtig abgespeichert werden.

Solche Operationen brauchen oft sehr viel Zeit und viele Taktzyklen, daher kommt häufig eine andere Strategie zum Einsatz:
Die CPU belegt die Register mit den korrekten Werten und aktiviert die entsprechenden Kontrollleitungen. Der Speicher meldet das Operationsende mit einem Signal an die Verarbeitungseinheit zurück. Diese Methode bezeichnet man als Hand-Shake-Verfahren. Statt einer sind bei modernen Architekturen zwei Leitungen üblich: Das Address-strobe-Signal, welches das Anliegen einer gültigen Adresse anzeigt, und das Data-Ready-Signal (oder Data Acknowledge), um das Ende der Operation bekannt zu geben.

Um die Schnittstelle benutzen zu können, werden folgende Funktionen benötigt:
- Laden der gewünschten Adresse in das Memory Address Register (MAR)
- Laden eines beliebigen Registerinhalts des Scratchpads in das Memory Buffer Register (MBR) vor der Speicheroperation
- Speichern des Inhalts des Memory Buffer Register (MBR) in einem Register des Scratchpads nach einer Leseoperation.



Durch den A-MUX lässt sich mit dem Signal A0 steuern, ob die Daten für die ALU aus dem A-Bus oder dem MBR kommen.
Durch die Aktivierung des Signals ldMAR wird das mit dem B-BUS verbundene MBR mit Daten befüllt. Die ldMBR-Leitung steuert – abhängig vom rd/wr-Signal – den Ladevorgang des MBR.



Mikro-Befehlssequenzen für das Lesen bzw. Schreiben:

	Micro-Instruction
Lesen(1)	$MAR \leftarrow RF_i$; rd
Lesen(2)	rd
Lesen(3)	$RF_k \leftarrow MBR$
Schreiben(1)	$MAR \leftarrow RF_i$; $MBR \leftarrow RF_j$; wr
Schreiben(2)	wr

Control Unit

Ein wichtiger Bestandteil ist der Clock-Baustein, der für die richtige Abfolge der Arbeitsschritte sorgt. Das bisher noch unbenannte Register, das als Speicher für die Mikro-Instruktion dient, bezeichnen wir ab jetzt mit MIR (Micro Instruction Register).

Bei unserer CPU (Mikro 16) fehlt noch eine wichtige Einheit – die Control Unit – mit folgenden Aufgaben:

1. Einlesen des nächsten Befehls in Micro-Code
2. Decodieren des Befehls, Laden des MIR mit den Daten
3. Anlegen der Steuersignale in zeitlich festgelegter Reihenfolge an die anderen Komponenten

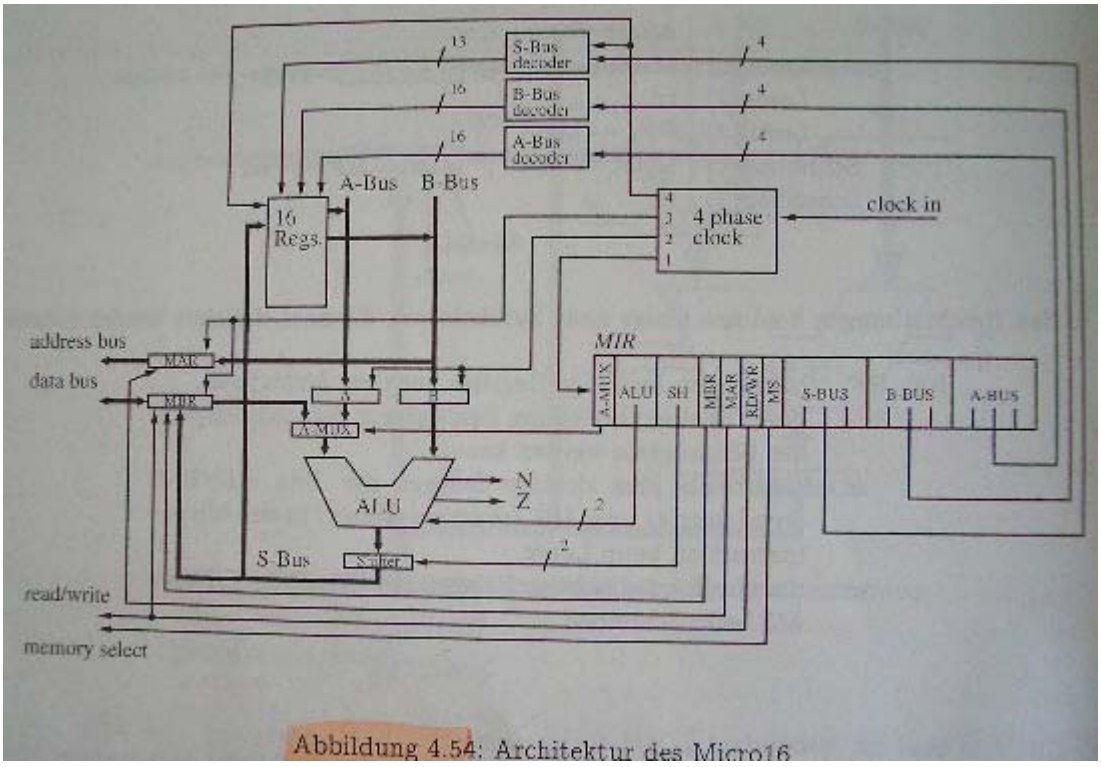


Abbildung 4.54: Architektur des Micro16

Es fehlt noch die Vorrichtung, in der sich die Mikro-Programme befinden, und eine Möglichkeit, sie in das MIR zu laden. Als speicher dient ein Micro-Code-ROM. Zum Adressieren eines Befehls findet ein Binärzähler mit 8 Stellen Verwendung, diesen nennt man MIC (Micro Instruction Counter). Der adressierte Befehl (dessen Adresse gleich dem Wert des Zählers ist) wird über parallele Leitungen in das MIR geladen. Wenn man den Zähler nach jedem Instruction Fetch um eins erhöht, kann mit dieser Anordnung ein sequenzielles Programm abgearbeitet werden.
Es fehlt jedoch noch die Möglichkeit für bedingte und unbedingte Sprünge, das Modell wird noch um diese Funktionen erweitert.

Für die Durchführung eines Sprungbefehls muss der MIC mit einem neuen Wert geladen werden. Daher ergänzt man den MIC um eine Parallel-Load-Funktion. Wir erweitern den Mikro-Befehl und das MIR um 8 Bit.

Folgende Sprungvarianten stehen zur Verfügung:

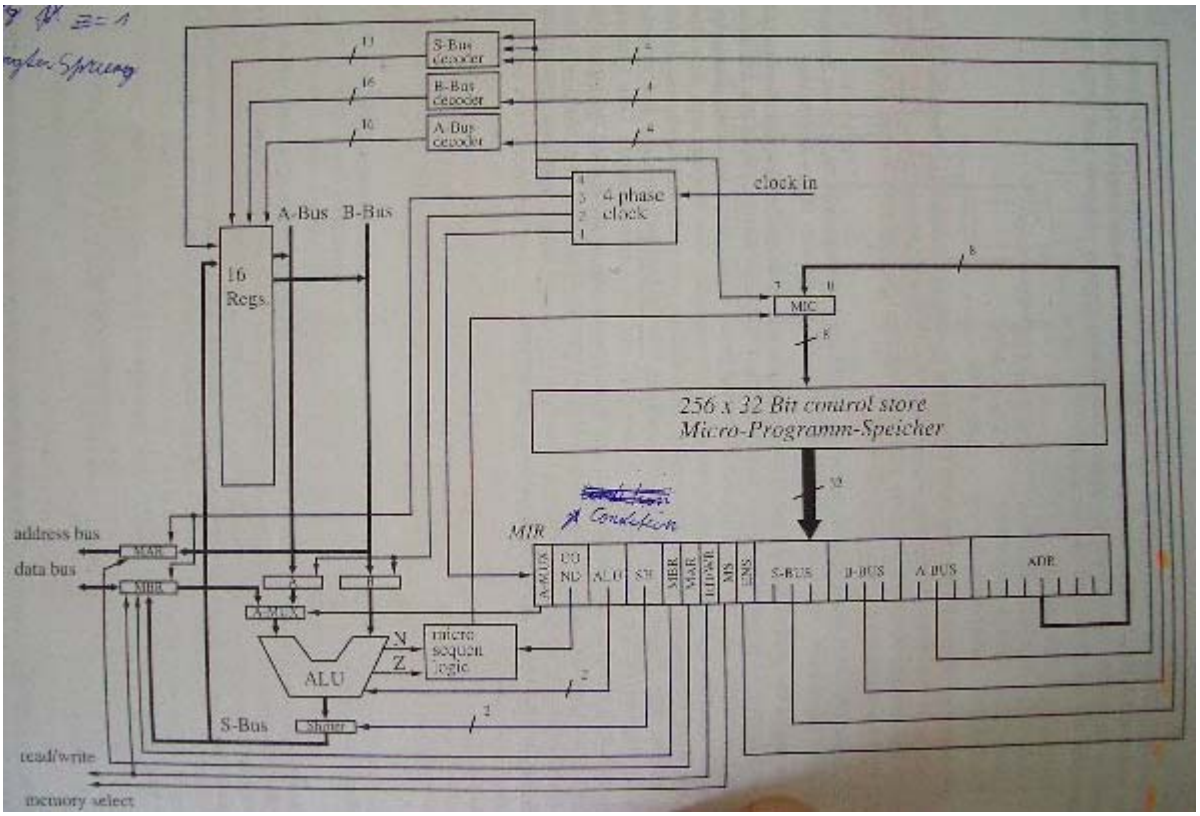
1. Sprung, wenn N = 1 gilt (ALU-Ergebnis < 0)
2. Sprung, wenn Z = 1 gilt (ALU-Ergebnis = 0)
3. Unbedingter Sprung

Um diese Fälle im Code abzubilden, benötigt man zwei weiter – als COND bezeichnete – Bits.

Cond=(00) kein Sprung
Cond=(01) Sprung, wenn N=1 if N goto ADR
Cond=(10) Sprung, wenn Z=1 if Z goto ADR
Cond=(11) unbedingter Sprung goto ADR

Oftmals ist es notwendig, für einen bedingten Sprung den Inhalt eines Registers oder die Verknüpfung zweier Register zu überprüfen, ohne das Ergebnis abzuspeichern. Dafür benutzt man ein zusätzliches Kontrollbit in der Mikro-Instruktion (genannt ENS Enable S-Bus). Der S-Bus wird freigegeben, wenn es gesetzt ist.

Rechnerarchitektur des Mikro-16:



Mikro Programm

Wir versuchen nun die Funktionalität des Mikro 16 zu steigern und schreiben ein Mikro Programm zum multipizieren von zwei ganzen Zahlen.

```
1.  R7←lsh(1+1)     # im ersten Schritt wird in R7 mit 1+1=2=(10)2
2.                  # und mit lsh((10)2=(100)2=(4)10 erzeugt
3.  R7←R7+R7        # R7=4+4=8
4.  R8←0            # Null ins Ergebnisregister laden
```

```
5.  R6←(1+1)          # R6=(2)10
6.  R6←R6+1           # R6=2+1=(3)10
7.  MAR←R6; rd        # Laden des ersten Operanden von der Speicherzelle (3)10
8.  rd                # warten, bis der Wert anliegt
9.  R9←MBR            # Speichern des Operanden in R9
10. R6←R6+1           # R6=3+1=(4)10 (Adresse des zweiten Operanden)
11. MAR←R6; rd        # Laden des zweiten Operanden von der Speicherzelle (4)10
12. rd                # warten, bis der Wert anliegt
13. R10←MBR           # Speichern des Operanden in R10
14. R9←lsh(R9+R9)     # dies entspricht zwei Shift-Kommandos nach links
15. R9←lsh(R9+R9)
16. R9←lsh(R9+R9)
17. R9←lsh(R9+R9)     # insgesamt 8 Shift-Operationen: R9=xxxx xxxx 0000 0000
18. (R7); if Z goto 28    # Test, ob (R7= 0); wenn ja Sprung auf Zeile 24
19.                       # bloßer Test ohne Speicherung: PNS= 0
20. R8←lsh(R8)            # Ergebnis um eine Stelle nach links shiften
21. (R9); if N goto 25    # wenn das msb im Multiplikator R9 0 ist, wird die
22.                       # folgende Zeile übersprungen
23. R8←R8+R10             # wenn das msb im Multiplikator R9 1 ist, wird der
24.                       # Multiplikant R10 zum Ergebnis R8 addiert
25. R9←lsh(R9)            # nächstes Bit des Multiplikators ins msb schieben
26. R7←R7-1               # Zähler R7 um 1 vermindern
27. goto 18               # unbedingter Sprung an den Schleifenbeginn
28. R6←R6+R6          # R6=4+4=(8)10
29. R6←R6+1           # R6=8+1=(9)10
30. MAR←R6            # Schreiben des Ergebnisses R8 auf die Speicherstelle (9)10
31. MBR←R8; wr
32. wr                # warten, bis der Wert verarbeitet ist
```

Very Large Scale Integration (VLSI)

Ein Mikro-Prozessor vereinigt alle Einzelschaltungen auf einem einzigen Chip. Nach dem Chipdesign werden alle Bauteile und Verbindungsleitungen gleichzeitig auf einem Siliziumplättchen (einem Halbleitermaterial) hergestellt. Diese Technik bezeichnet man als Very Large Scae Integration (VLSI).
Heute besteht die Tendenz, möglichst viele Bauteile auf einem Chip zu integrieren. Da sich der elektrische Strom nur mit 0,7 facher Lichtgeschwindigkeit ausbreitet, bedeutet die reduzierung der Wegstrecken eine Leistungssteigerung, jedoch muss dann auch mehr Wärme abgeführt werden.