

Skriptum zur Vorlesung

Grundlagen der Informationsverarbeitung II

Sommersemester 2004

Betreuer:

Prof. Dr.-Ing. H. Schröder

Raum: P1-04-211

Tel.: 755-3190

Email: hartmut.schroeder@uni-dortmund.de

Dipl.-Ing. H. Lange

Raum: P1-04-210

Tel.: 755-2028

Email: hendrik.lange@uni-dortmund.de

Aktuelle Informationen zur Vorlesung gibt es unter:

http://www-kt.e-technik.uni-dortmund.de/m_siv/giv/giv_main.html

Inhaltsverzeichnis

Kapitel 1 Einführung	1
1.1 Inhaltsübersicht	1.1
1.2 Formen informationsverarbeitender Schaltungen und Systeme (mit Programm-steuerung)	1.2
1.3 Ziel der Vorlesung	1.4
1.4 Literaturhinweise zur Vorlesungsbegleitung:	1.5
 Kapitel 2 Schaltnetze	 2.1
2.1 Boolesche Algebra	2.1
2.1.1 Formale Definition der Booleschen Algebra	2.1
2.1.2 Formale Definition der Schaltalgebra	2.2
2.1.3 Boolesche Ausdrücke, Definitionen	2.3
2.1.4 Boolesche Funktionen	2.3
2.1.5 Rechenregeln bei Booleschen Ausdrücken	2.6
2.1.6 Minterme, Maxterme	2.7
2.1.7 Disjunktive Normalform	2.8
2.1.8 Konjunktive Normalform (KNF)	2.8
2.1.9 Beispiele für disjunktive und konjunktive Formen	2.9
2.1.10 Partiell definierte Boolesche Funktionen	2.11
2.2 Logische Schaltungen, Schaltnetz und Schaltwerk	2.13
2.2.1 Schaltwerke	2.14
2.2.2 Schaltnetze	2.14
2.2.3 Gatterschaltungen	2.14
2.2.4 Schaltungen mit nur einem Gattertyp	2.15
2.3 Systematische Minimierungsverfahren für Schaltnetze	2.16
2.3.1 Karnaugh-Veitch (KV)-Diagramme	2.17
2.3.1.1 Boolescher Funktionen in KV-Diagrammen	2.20
2.3.1.2 Minimierung mit KV-Diagrammen	2.25
2.3.2 Minimierungsverfahren nach Quine-McCluskey	2.26
2.4 Ausgewählte Beispiele zu logischen Schaltungen	2.30
2.4.1 Zweistelliger Dualzahlenvergleicher	2.30
2.4.2 1-aus-N-Decoder, Prioritäts-Encoder	2.32
2.4.3 Multiplexer, Demultiplexer	2.34
2.4.4 Schaltnetze mit Multiplexern	2.35
2.5 CMOS- Grundsaltungen	2.38

2.5.1.1	Transmission-Gate	2.42
2.5.1.2	Multiplexer	2.43
2.5.2	Literaturhinweise	2.44
Kapitel 3	Arithmetisch-logische Einheiten	3.1
3.1	Digitale Zeichen- und Zahlendarstellungen	3.1
3.2	Digitale Zahlendarstellungen, Festkomma- und Ganzzahlenformate	3.2
3.2.1	Das Betrag-Vorzeichen-Format	3.4
3.2.2	Das Zweierkomplement-Format	3.5
3.3	Addierer-Schaltnetze	3.9
3.3.1	Addierer-Schaltnetze mit Übertragsweiterleitung	3.10
3.3.2	Addiererschaltnetz mit Übertragsvorausberechnung	3.14
3.3.3	Akkumulation von Dualzahlen	3.17
3.3.4	Rechenwerk zur Addition und Subtraktion von 2K-Zahlen	3.20
3.3.5	Arithmetisch-logische Einheiten	3.24
3.4	Schaltwerke zur binären Multiplikation	3.26
3.4.1	Seriell-parallele Multiplikation in BV-Darstellung	3.27
3.4.2	Paralleles Multiplikationswerk	3.30
Kapitel 4	Schaltwerke	4.1
4.1	Begriff und Funktion des Schaltwerkes	4.1
4.2	Elementare Schaltwerke (Register und Flipflops)	4.2
4.2.1	SR-Flip Flop (SR-FF) zur Binärsignalspeicherung	4.2
4.2.1.1	Zustandsgraphen	4.4
4.2.1.2	Zustandsgraph, Übergangstabelle und -gleichung	4.5
4.2.2	Taktgesteuerte Flip Flops	4.7
4.2.2.1	Taktzustandsgesteuertes SR-Flipflop	4.7
4.2.2.2	Taktzustandsgesteuertes D-Flipflop	4.8
4.2.2.3	Taktflankengesteuertes SR-Flipflop	4.9
4.2.2.4	Taktflankensteuerung durch Zwischenspeicher-FF	4.10
4.2.3	Master-Slave-Technik zur Zweizustandssteuerung	4.12
4.2.3.1	SR-Master-Slave-FF, 2-zustandsgesteuert	4.13
4.2.3.2	JK-Master-Slave-FF, 2-zustandsgesteuert	4.14
4.2.3.3	MS-FF mit Taktzustands- und Taktflankensteuerung	4.16
4.2.3.4	Toggle-MS-Flip Flop	4.19
4.2.3.5	D-Master-Slave-FF	4.19
4.2.4	Arten zweiflankengesteuerter Flip Flops	4.20
4.3	Formale Schaltwerkbeschreibung	4.22
4.3.1	Asynchrone Schaltwerke	4.24

4.3.2	Synchrone Schaltwerke	4.27
4.3.2.1	Frequenzteiler (Modulo-4-Zähler)	4.27
4.3.2.2	Extern einstellbares Modulo-2-Zählwerk	4.29
4.3.2.3	Ein Modulo4-Steuerwerk	4.32
4.3.2.4	Ein Modulo 8-Zählwerk aus Toggle Flip Flops	4.36
4.4	Allgemeine Hinweise zum Schaltwerkentwurf	4.37
4.5	Verbindungen zur Automatentheorie nach [LiPP]	4.38
4.6	Prinzip der Mikroprogrammierung	4.46
4.6.1	Mikroprogrammsteuerwerk	4.48
4.6.2	Prinzip einer Register-ALU (RALU)	4.49
4.6.3	Mikroprogrammierbare Rechner (Firmware)	4.51
Kapitel 5	Speicherstrukturen und logische Felder	5.1
5.1	Halbleiterspeicher	5.1
5.1.1	Speicher mit bitweiser Organisation	5.6
5.1.2	Speicher mit wortweiser Organisation	5.7
5.1.3	Adressdecoder zur Speicheradressierung	5.8
5.2	Logische Beschreibung von Festwertspeichern	5.9
5.3	Programmierbare logische Felder	5.11
5.3.1	Tabellen mit Hilfe von Festwertspeichern und log. Feldern	5.14
5.4	Übersicht: Anwenderprogrammierbare Schaltungen und log. Felder	5.17
5.4.1	Einordnung bei den integrierten Schaltungen	5.17
5.5	FPLD: Field Programmable Logic Device – Übersicht	5.21
5.6	EPROM / Complex Programmable Logic Devices (EPLDs, CPLDs)	5.22
5.6.1	CPLD-Bausteine, Beispiel: Altera 7000 EPLD	5.25
5.7	Antifuse-FPGA, Beispiel: ACTEL-Bausteine	5.28
5.8	SRAM-basierte Bausteine, Beispiel: Xilinx-Bausteine	5.29
5.9	Bemerkungen zum Entwurf programmierbarer Bausteine	5.31
5.9.1	Entwurfswerkzeuge	5.32

Kapitel 1

Einführung

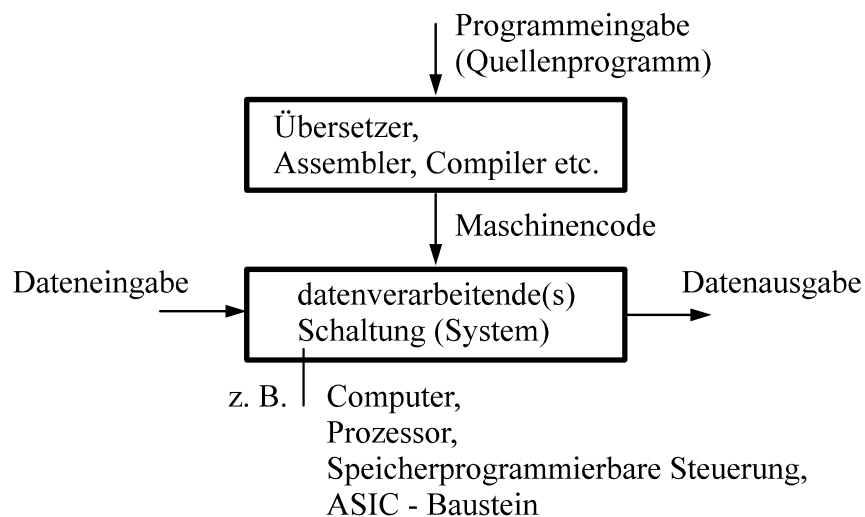
1.1 Inhaltsübersicht

Das Skript zur Vorlesung „Grundlagen der Informationsverarbeitung 2“ gliedert sich in folgende Kapitel:

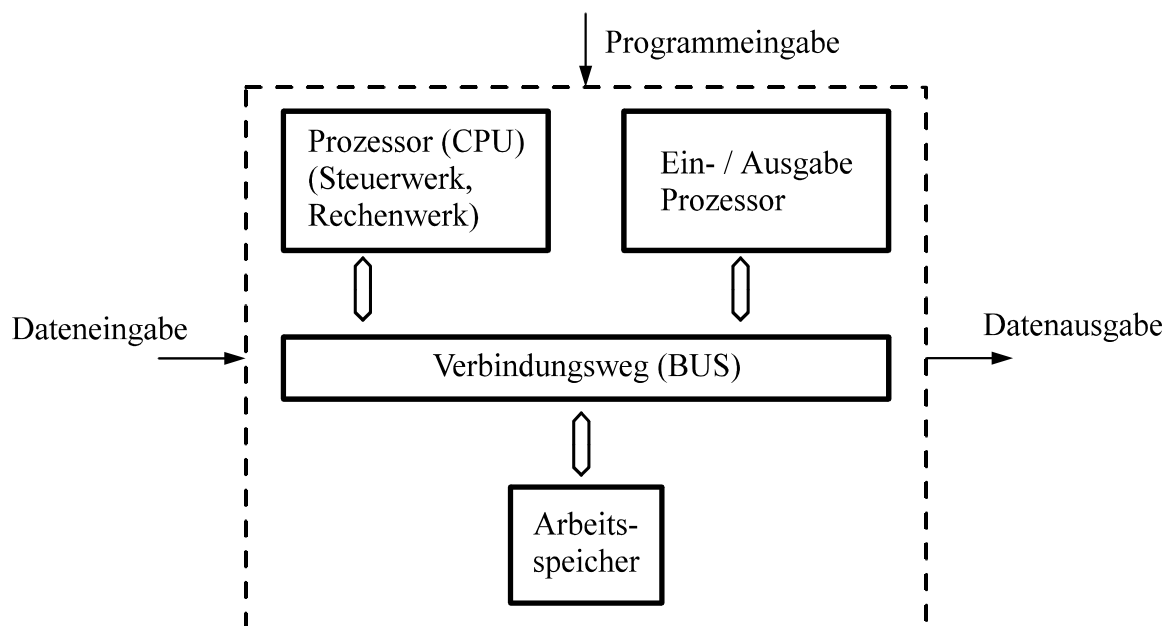
- 1 Einführung** Ziel der Vorlesung, Formen informationsverarbeitender Schaltungen, Bedeutung, geschichtliche Entwicklung, Inhalte der Vorlesung, Literaturhinweise
- 2 Schaltnetze** Boolesche Algebra, Boolesche Ausdrücke und Funktionen, kanonische Darstellung, Strukturen von Schaltnetzen, algebraische, graphische und algorithmische Minimierung, Bausteinfamilien für logische Schaltungen, Datenwegschaltungen, CMOS-Grundsaltungen
- 3 Arithmetisch-logische Einheiten** Schaltketten, Zahlendarstellungen, Addition und Subtraktion von Dualzahlen, Addiererschaltnetze, Übertragungstechniken, arithmetisch-logische Einheiten (ALU), Funktionseinheiten zur Multiplikation
- 4 Schaltwerke** Darstellungsformen (Übertragungsfunktion, -tabelle, -graphen), asynchrone und synchrone Schaltwerke, elementare Schaltwerke (Flip-Flops), formale Beschreibung, Schaltwerke und Automaten, Schaltwerke zur Informationsspeicherung (Register), Zähler, Steuerwerke, Leitwerke, Mikroprogrammierung, Register-ALU
- 5 Logische Felder und anwenderprogrammierbare Schaltungen** Klassen integrierter Schaltungen, Festwertspeicher, programmierbare logische Felder, FPLD: field programmable logic device, FPGA: field programmable gate array, Bausteinfamilien (Beispiele), Hinweise zum Entwurf.

1.2 Formen informationsverarbeitender Schaltungen und Systeme (mit Programmsteuerung)

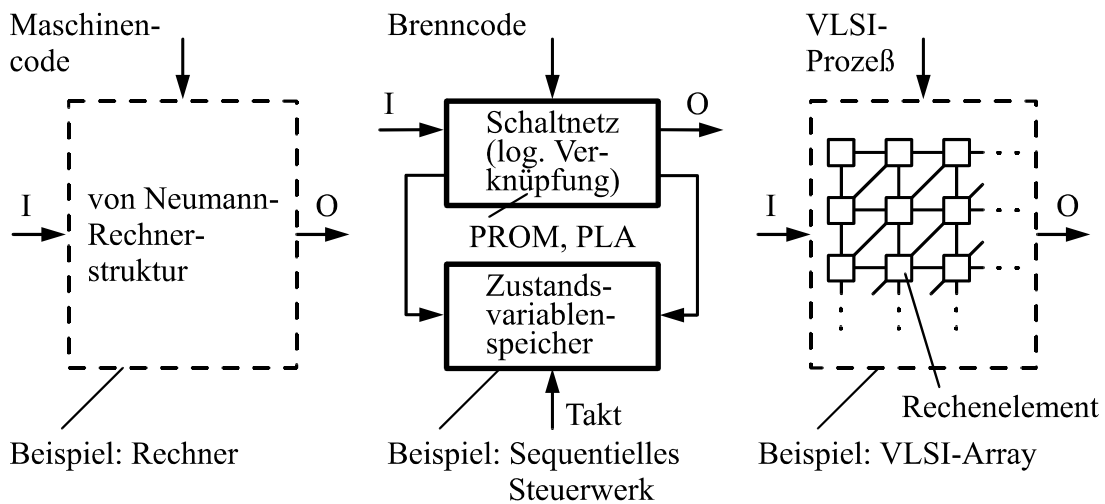
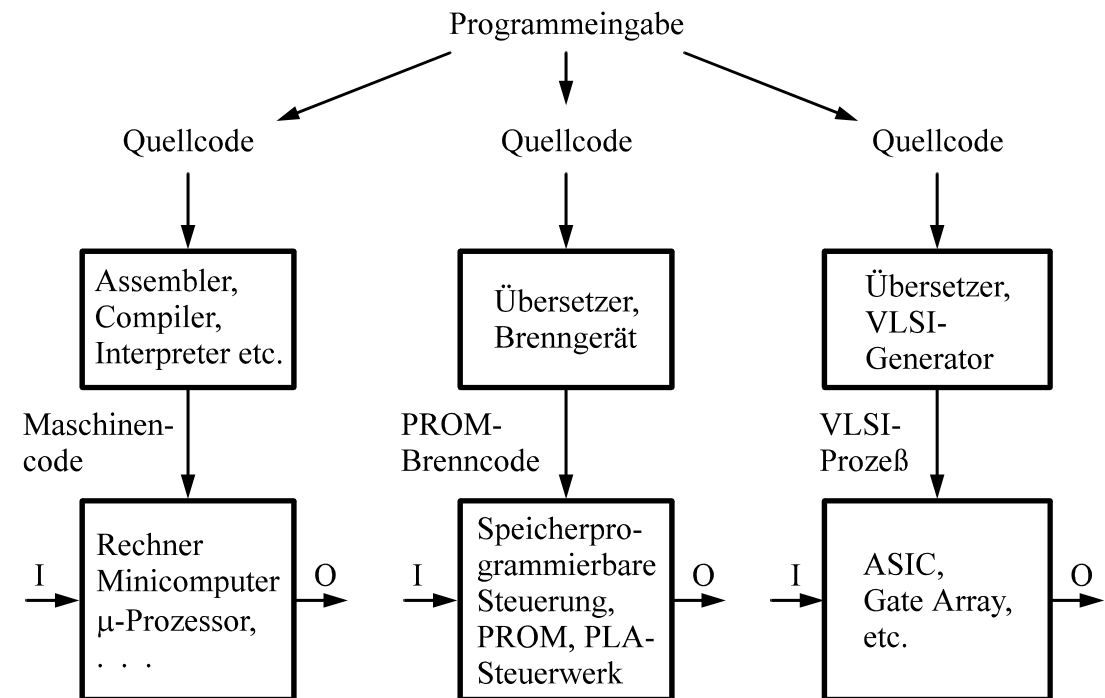
Allgemein:



Klassische Von-Neumann-Rechner-Struktur:



Realisierungsformen informationsverarbeitender Schaltungen, Beispiele



1.3 Ziel der Vorlesung

Die Vorlesung „Grundlagen der Informationsverarbeitung II“ behandelt die Grundzüge logischer Schaltungen, d.h. die Grundlagen datenverarbeitender Schaltungen und Systeme.

Ziel ist die Vermittlung von Kenntnissen zu

- (1) Grundlagen datenverarbeitender Schaltungen
 - Schaltnetze, Schaltketten
 - Schaltwerke (sequentielle Schaltungen)
- (2) Elementare Schaltungen
 - Logikschaltungen
 - Arithmetikbausteine
 - Register, Speicherkomponenten
- (3) Komplexe Schaltungen und integrierte Bausteine
 - Arithmetisch-logische Einheiten, Steuerwerke
 - Logische Felder
 - Anwenderprogrammierbare integrierte Schaltungen

1.4 Literaturhinweise zur Vorlesungsbegleitung:

Schiffmann et al	Technische Informatik Band 1 und 2, Springer Verlag, 1999, (Grundlagen, Anwendungsorientiert)
Lipp, Hans Martin	Grundlagen der Digitaltechnik, Oldenbourg Verlag, 1998, (theoretisch fundierte Einführung)
Scarbata, Gerd	Synthese und Analyse digitaler Schaltungen, Oldenbourg, 1996, (Einführung mit Schwerpunkt Entwurfmethodik)
Eschermann, B	Funktionaler Entwurf digitaler Schaltungen, Springer Verlag, 1993, (Einführung, anwendungsorientiert mit technologischem Bezug)
Schröder, Hartmut, Blume, Holger	Mehrdimensionale Signalverarbeitung Band II, Teubner, 2000 (Schwerpunkt Signalverarbeitung mit einer grundlagenorientierten Hinführung)
Salcic, Z. et al	Digital Systems Design and Prototyping using Field Programmable Logic, Kluwer, 1998, (Programmierbare logische Schaltungen)
Dueck, R. K.	Digital Design with CPLD Applications and VHDL, Delmar Thomson Learning, 2001, (Programmierbare logische Schaltungen)
Clements, S.	The Principles of Computer Hardware, Oxford, 2000, (spezifisch für arithmetisch-logische Schaltungsarchitekturen)
U. Tietze, Ch. Schenk	Halbleiter-Schaltungstechnik, Springer Verlag, 1999, (anwendungsorientierte Darstellung bei Verzicht auf eine Darstellung der Grundlagen)

Kapitel 2

Schaltnetze

2.1 Boolesche Algebra

2.1.1 Formale Definition der Booleschen Algebra

(Boole, 1854) durch das Axiomensystem von Huntington (1904): Gegeben sei eine Menge M mit den Elementen $a_1, a_2, \dots$. Zwischen den Elementen der Menge seien zweistellige Operationen Φ und Ψ derart definiert, daß durch Anwendung der Operationen wiederum eindeutig Elemente von M entstehen. Es gelten im übrigen folgende Axiome A1 bis A4:

A1 Kommutativgesetze

$$a_1 \Phi a_2 = a_2 \Phi a_1$$

$$a_1 \Psi a_2 = a_2 \Psi a_1$$

A2 Distributivgesetze

$$a_1 \Phi (a_2 \Psi a_3) = (a_1 \Phi a_2) \Psi (a_1 \Phi a_3)$$

$$a_1 \Psi (a_2 \Phi a_3) = (a_1 \Psi a_2) \Phi (a_1 \Psi a_3)$$

A3 Neutrale Elemente, Einselemente

Es existieren für jede der beiden Operationen Φ und Ψ je ein Einselement e und n , für welche die Verknüpfung mit einem beliebigen Element $a_i \in M$ wiederum a_i ergibt.

$$a_i \Phi e = a_i \quad \forall a_i \in M$$

$$a_i \Psi n = a_i \quad \forall a_i \in M$$

A4 Komplement

Zu jedem Element $a_i \in M$ existiert auch das Komplement $\bar{a}_i \in M$, so dass gilt:

$$a_i \Phi \bar{a}_i = n$$

$$a_i \Psi \bar{a}_i = e$$

Beispiele für eine Boolesche Algebra

- Mengenalgebra: Operation Φ : Durchschnitt
Operation Ψ : Vereinigung

- Aussagenalgebra/Schaltalgebra: Algebra für Mengen mit 2 Elementen $M = \{0,1\}$

e ersetzen durch 1 (wahr)

n ersetzen durch 0 (falsch)

Φ ersetzen durch $\wedge$ (logisches UND, Konjunktion)

Ψ ersetzen durch $\vee$ (logisches ODER, Disjunktion).

Eine Aussage (Schalterzustand) wird durch zwei Teile beschrieben, einen Namen zur Kennzeichnung der Aussage (des Schalters) und einen Wert (Zustand) als Inhalt der Aussage (Angabe der Schalterstellung). Man spricht von einer Aussagenvariablen bzw. Schaltvariablen (logische Variable).

2.1.2 Formale Definition der Schaltalgebra

Shannon (1938): Gegeben ist eine Menge $M = \{0,1\}$ und die beiden Operationen UND ($\wedge$) und ODER ($\vee$). Es gelten im übrigen die Axiome A1-A4 (a, b, c: logische Variablen):

A1 Kommutativgesetze:

$$a \wedge b = b \wedge a$$

$$a \vee b = b \vee a$$

A2 Distributivgesetze:

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) = a \wedge b \vee a \wedge c$$

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

Zur Vereinfachung der Schreibweise wird verabredet, daß UND ($\wedge$) vor ODER ($\vee$) ausgeführt wird.

A3 Einselement:

$$a \wedge 1 = a$$

$$a \vee 0 = a$$

A4 Komplement:

$$a \wedge \bar{a} = 0$$

$$a \vee \bar{a} = 1$$

Zur Vereinfachung der Schreibweise wird verabredet, daß die Komplementierung $\bar{}$ vor den anderen Operationen ausgeführt wird.

2.1.3 Boolesche Ausdrücke, Definitionen

Gegeben sei die Menge der Variablen $X = \{x_1, x_2, \dots, x_n\}$. Die Variablen können die Werte 0 oder 1 annehmen und werden in Booleschen Ausdrücken miteinander verknüpft. Das heißt, Boolesche Ausdrücke sind aus den Variablen $x_i \in X$, den Konstanten 0 oder 1, den Operationszeichen $\wedge, \vee, \neg$ und den Klammersymbolen zusammengesetzt. Werte für Boolesche Ausdrücke sind wiederum 0 bzw. 1. Mit solchen Booleschen Ausdrücken lassen sich logische Kombinationsschaltungen beschreiben. Es gilt:

- (1) 0 und 1 sind Boolesche Ausdrücke
- (2) Jede Variable x_i ist Boolescher Ausdruck
- (3) Ist A ein Boolescher Ausdruck, dann ist auch $\overline{A}$ ein Boolescher Ausdruck
- (4) A und B seien Boolesche Ausdrücke, dann sind auch $(A \vee B)$ und $(A \wedge B)$ Boolesche Ausdrücke
- (5) Die Menge A_N der Booleschen Ausdrücke in den Variablen $x_1, x_2, \dots, x_n$ wird durch die Regeln (1) - (4) beschrieben.

2.1.4 Boolesche Funktionen

Die Auswertung eines Booleschen Ausdruckes erfolgt stets für eine bestimmte Kombination von Werten b_i der Variablen X (Belegung).

$$x_i = b_i \text{ mit } i = 1, 2, \dots, n, \quad b_i \in \{0, 1\}$$

Für n Variable x_i existieren genau $m = 2^n$ verschiedene Belegungen (0,1-Muster).

Durch den Ausdruck

$$A(b_1, b_2, \dots, b_n) = f$$

wird jedem n-Tupel $(b_1, b_2, \dots, b_n), b_i \in \{0, 1\}$, d.h. jeder Belegung, ein Wert aus $\{0, 1\}$ zugeordnet. Ein Boolescher Ausdruck definiert somit eine Abbildung f der Menge $\{0, 1\}^n$ der n-Tupel auf die Menge $\{0, 1\}$. Eine solche Abbildung wird als n-stellige Boolesche Funktion oder logische Funktion bezeichnet.

$$y = f(x_1, x_2, \dots, x_n) \quad x_i, y \in \{0, 1\}$$

Ein Boolescher Ausdruck definiert also eine Boolesche Funktion der in ihm auftretenden Variablen. Ein Boolescher Ausdruck bzw. eine Boolesche Funktion läßt sich auch durch eine Wertetabelle beschreiben.

Jede n-stellige Boolesche Funktion hat nur endlich viele Argumente, dies sind die 2^n Kombinationen der Werte der n Variablen. Für diese 2^n Wertekombinationen einer

Funktion sind wiederum 2^{2^n} Kombinationen aus $\{0,1\}$ denkbar, d.h. es gibt 2^{2^n} nicht äquivalente (verschiedene) n -stellige Boolesche Funktionen.

Für eine und zwei Variablen lassen sich die möglichen booleschen Funktionen noch leicht überblicken und benennen:

Eine Variable:

x	f ₀	f ₁	f ₂	f ₃
0	0	1	0	1
1	0	0	1	1

f₀=0 Nullfunktion (z.B. $x \bar{x}$)

f₁= $\bar{x}$ Negation

f₂= x Identität

f₃=1 Einsfunktion (z.B. $x \vee \bar{x}$)

Zwei Variable:

a	b	f ₀	f ₁	f ₂		f ₁₅
0	0	0	0	.	.	1
1	1	0	0	.	.	1
0	1	0	0	.	.	1
0	1	0	1	.	.	1

Für die Verknüpfung zweier Variablen gibt es also 16 mögliche verschiedene Funktionen, die in der folgenden Tabelle zusammengefaßt sind. Ersichtlich lassen sich alle zweistelligen Funktionen durch Boolesche Ausdrücke mit höchstens zwei Variablen darstellen.

Werte- tabelle a 0101 b 0011	Boolesche Funktion Verknüpfungstabelle		Bezeichnung Aussagen- algebra	alternative Bezeichnung verbale Beschreibung	andere Ver- knüpfungs- symbole	Venn- diagramme a·b
0 0 0 0	$f_0 = 0$		Kontradiktion	Nullfunktion (Nie)		
0 0 0 1	$f_1 = a \wedge b$		Konjunktion	UND	$a \cdot b, a \& b$	
0 0 1 0	$f_2 = \bar{a} \wedge b$	$b \rightarrow a$	Inhibition	wenn b dann $\bar{a}$		
0 0 1 1	$f_3 = b$		Identität			
0 1 0 0	$f_4 = a \wedge \bar{b}$	$a \rightarrow b$	Inhibition	wenn a dann $\bar{b}$		
0 1 0 1	$f_5 = a$		Identität			
0 1 1 0	$f_6 = (a \wedge \bar{b}) \vee (\bar{a} \wedge b)$	$a \neq b$	Antivalenz	Exklusiv ODER	$a \oplus b$	
0 1 1 1	$f_7 = a \vee b$		Disjunktion	ODER	$a + b$	
1 0 0 0	$f_8 = \overline{a \vee b}$		Peircefunktion	NOR	$a \downarrow b, a * b$	
1 0 0 1	$f_9 = (a \vee \bar{b}) \wedge (\bar{a} \vee b)$	$a \equiv b$	Äquivalenz	gleichwertig	$a \leftrightarrow b$	
1 0 1 0	$f_{10} = \bar{a}$		Negation		- a	
1 0 1 1	$f_{11} = \bar{a} \vee b$	$a \rightarrow b$	Implikation	Wenn a dann b	$a \supset b$	
1 1 0 0	$f_{12} = \bar{b}$		Negation		- b	
1 1 0 1	$f_{13} = a \vee \bar{b}$	$b \rightarrow a$	Implikation	Wenn b dann a	$b \supset a$	
1 1 1 0	$f_{14} = \overline{a \wedge b}$		Shefferfunktion	NAND	$a b$	
1 1 1 1	$f_{15} = 1$		Einsfunktion	Tautologie (immer)		

Bild 2.1: Zweistellige boolesche Funktionen

2.1.5 Rechenregeln bei Booleschen Ausdrücken

<u>Kommutativgesetz</u>	$A \wedge B = B \wedge A$ $A \vee B = B \vee A$
<u>Distributivgesetze</u>	$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C) = A \wedge B \vee A \wedge C$ $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
<u>Einselement</u>	$A \wedge 1 = A$ $A \vee 0 = A$
<u>Komplementbildung</u>	$A \vee \overline{A} = 1$ $A \wedge \overline{A} = 0$

Hieraus lassen sich ableiten:

<u>Assoziativgesetze</u>	$A \vee (B \vee C) = (A \vee B) \vee C$ $A \wedge (B \wedge C) = (A \wedge B) \wedge C$
<u>Gesetz von de Morgan</u>	$\overline{A \vee B} = \overline{A} \wedge \overline{B}$ $\overline{A \wedge B} = \overline{A} \vee \overline{B}$
<u>Absorptionsgesetze</u>	$A \wedge (A \vee B) = A$ $A \vee (A \wedge B) = A$
<u>Idempotenzgesetze</u>	$A \vee A = A$ $A \wedge A = A$
<u>Reduktionsformeln</u>	$A \vee (\overline{A} \wedge B) = (A \vee \overline{A}) \wedge (A \vee B) = A \vee B$ $A \wedge (\overline{A} \vee B) = (A \wedge \overline{A}) \vee (A \wedge B) = A \wedge B$

Prioritätsvereinbarung: " Negation vor Konjunktion vor Disjunktion "

(Häufig wird $\wedge$ weggelassen: $A \wedge B = AB$)

Dualitätsprinzip: Jede Aussage, die aus den Axiomen einer Booleschen Algebra hergeleitet ist, ist gültig bzw. liefert eine gültige boolesche Aussage, wenn die Operatoren $\wedge$ und $\vee$ und die Elemente 0 und 1 gegeneinander an allen Stellen vertauscht werden. Die beiden Aussagen, die neue und die, von der ausgegangen wurde, sind „dual“ zueinander.

2.1.6 Minterme, Maxterme

Ein Minterm M_k^i (Vollkonjunktion) von n Variablen ist eine konjunktive Verknüpfung (Boolesches Produkt) dieser n Variablen, worin jede dieser n Variablen genau einmal negiert oder nicht negiert enthalten ist. Es gibt bei n binären Variablen insgesamt 2^n Minterme:

$$M_k^i = (a_1 \equiv b_1) \wedge (a_2 \equiv b_2) \wedge \dots \wedge (a_n \equiv b_n) = \bigwedge_{j=1}^n (a_j \equiv b_j)$$

$$b_j \in \{0, 1\}, k=0, 2, \dots, 2^n$$

Ein Maxterm M_k^a (Volldisjunktion) ist die entsprechende disjunktive Verknüpfung (Boolesche Summe) der n Variablen (vgl. Dualitätsprinzip):

$$M_k^a = (a_1 \equiv \bar{b}_1) \vee (a_2 \equiv \bar{b}_2) \vee \dots \vee (a_n \equiv \bar{b}_n) = \bigvee_{j=1}^n (a_j \equiv \bar{b}_j)$$

$$\bar{b}_j \in \{0, 1\}, k=0, 2, \dots, 2^n$$

Das n -Tupel $B_k = (b_1, b_2, \dots, b_n)$ gibt die Belegung der Variablen mit $(0,1)$ -Werten an. Jeder möglichen Belegung ist ein Minterm und Maxterm zugeordnet. Für drei Variablen gilt:

a b c	Minterm	Maxterm	Erläuterung
0 0 0	$\bar{a} \wedge \bar{b} \wedge \bar{c}$	$a \vee b \vee c$	<u>Minterm:</u> Boolesche Funktion nimmt für minimal wenige Terme den Wert 1 an, für nämlich genau einen; <u>Maxterm:</u> Boolesche Funktion nimmt für maximal viele Terme den Wert 1 an, nämlich für alle außer einem.
0 0 1	$\bar{a} \wedge \bar{b} \wedge c$	$a \vee b \vee \bar{c}$	
0 1 0	$\bar{a} \wedge b \wedge \bar{c}$	$a \vee \bar{b} \vee c$	
0 1 1	$\bar{a} \wedge b \wedge c$	$a \vee \bar{b} \vee \bar{c}$	
1 0 0	$a \wedge \bar{b} \wedge \bar{c}$	$\bar{a} \vee b \vee c$	
1 0 1	$a \wedge \bar{b} \wedge c$	$\bar{a} \vee b \vee \bar{c}$	
1 1 0	$a \wedge b \wedge \bar{c}$	$\bar{a} \vee \bar{b} \vee c$	
1 1 1	$a \wedge b \wedge c$	$\bar{a} \vee \bar{b} \vee \bar{c}$	

2.1.7 Disjunktive Normalform

Unter einer **disjunktiven Form (DF)** einer Booleschen Funktion $f(a_1, a_2, \dots, a_n)$ versteht man einen Ausdruck, der aus disjunktiv verknüpften Konjunktionen K_i besteht:

$$f(a_1, a_2, \dots, a_n) = K_1 \vee K_2 \vee \dots \vee K_m$$

Eine Konjunktion K_i besteht aus negierten oder nicht negierten Variablen die ausschließlich konjunktiv verknüpft sind.

Unter einer **Disjunktiven Normalform (DNF)** versteht man diejenige disjunktive Form, deren Konjunktionen alle Variablen umfassen, d.h. die Minterme darstellen:

$$f(a_1, a_2, \dots, a_n) = M_1^i \vee M_2^i \vee \dots \vee M_k^i$$

Jede Boolesche Funktion lässt sich eindeutig durch ihre disjunktive Normalform darstellen. Es treten in der DNF genau die Minterme auf, bei denen die zugehörige Belegung mit (0,1)-Werten den Mintermwert 1 (und damit den Funktionswert 1) liefern. Es besteht damit die Möglichkeit eine als Wertetabelle gegebene Funktion als Booleschen Ausdruck anzugeben.

2.1.8 Konjunktive Normalform (KNF)

Unter einer **konjunktiven Form (KF)** einer Booleschen Funktion $f(a_1, a_2, \dots, a_n)$ versteht man einen Ausdruck, der aus konjunktiv verknüpften Disjunktionen besteht:

$$f(a_1, a_2, \dots, a_n) = D_1 \wedge D_2 \wedge \dots \wedge D_m$$

Eine Disjunktion besteht aus negierten oder nicht negierten Variablen, die ausschließlich disjunktiv verknüpft sind.

Eine **konjunktive Normalform (KNF)** ist diejenige konjunktive Form, deren Disjunktionen alle Variablen enthalten, d.h. Maxterme darstellen:

$$f(a_1, a_2, \dots, a_n) = M_1^a \wedge M_2^a \wedge \dots \wedge M_k^a$$

Wiederum lässt sich jede Boolesche Funktion eindeutig durch ihre KNF darstellen. Es treten gerade die Maxterme auf, bei denen die zugehörigen (0,1)-Belegungen den Maxtermwert bzw. den Funktionswert 0 liefern.

Die DF bzw. KF erhält man durch

- Ersetzen der Operatoren $\neq$, $\equiv$, ... durch $\wedge$, $\vee$ und Negation
- wiederholte Anwendung der Regeln von de Morgan bis Negationen nur über einzelnen Variablen auftreten
- wiederholte Anwendung des jeweiligen Distributivgesetzes

2.1.9 Beispiele für disjunktive und konjunktive Formen

Die DNF erhält man aus der DF durch konjunktives Hinzufügen bei jeder Konjunktion aller in dieser nicht auftretenden Variablen a_i in der Form $(a_i \vee \bar{a}_i)$ und anschließendem "ausmultiplizieren" (Distributivgesetz).

Die KNF erhält man aus der KF durch disjunktives Hinzufügen bei jeder Disjunktion aller in dieser nicht auftretenden Variable a_i in der Form $(a_i \wedge \bar{a}_i)$ und anschließendem "ausaddieren" (Distributivgesetz).

Hinweis: In der Literatur wird zum Teil die disjunktive Form auch disjunktive Normalform, und die konjunktive Form auch konjunktive Normalform genannt. Die hier mit Normalform bezeichneten Ausdrücke sind dann "ausgezeichnete" oder "kanonische" Normalformen.

Beispiele für eine KF:

$$f(a_1, a_2, a_3, a_4) = (a_1 \vee \bar{a}_2 \vee a_4) \wedge (\bar{a}_2 \vee \bar{a}_3 \vee a_4) \wedge (a_3 \vee \bar{a}_4)$$

Beispiel für eine KNF:

a_1	a_2	a_3	f	
0	0	0	0	$f = (a_1 \vee a_2 \vee a_3) \wedge$ $(a_1 \vee a_2 \vee \bar{a}_3) \wedge$
0	0	1	0	
0	1	0	1	
0	1	1	1	
1	0	0	0	$(\bar{a}_1 \vee a_2 \vee a_3) \wedge$
1	0	1	1	
1	1	0	0	$(\bar{a}_1 \vee \bar{a}_2 \vee a_3) \wedge$
1	1	1	0	

KNF erhält man durch konjunktive Verknüpfung der Maxterme, an denen $f = 0$ ist (vgl. Dualitätsprinzip).

Beispiel für eine DF:

$$\begin{aligned} f_{DF}(a_1, a_2, a_3, a_4) &= a_1 a_2 \bar{a}_4 \vee a_2 a_3 \bar{a}_4 \vee \underbrace{a_1 a_3 a_4 \vee a_3 a_4}_{=1} \\ &= a_1 a_2 \bar{a}_4 \vee a_2 a_3 \bar{a}_4 \vee a_3 a_4 \underbrace{(a_1 \vee 1)}_{=1} \end{aligned}$$

Beispiel für eine DNF

a_1	a_2	a_3	a_4	f	
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	1	$f_{DNF} = \bar{a}_1 \bar{a}_2 a_3 a_4 \vee$
0	1	0	0	0	
0	1	0	1	0	
0	1	1	0	1	$\bar{a}_1 a_2 a_3 \bar{a}_4 \vee$
0	1	1	1	1	$\bar{a}_1 a_2 a_3 a_4 \vee$
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	0	
1	0	1	1	1	$a_1 \bar{a}_2 a_3 a_4 \vee$
1	1	0	0	1	$a_1 a_2 \bar{a}_3 \bar{a}_4 \vee$
1	1	0	1	0	
1	1	1	0	1	$a_1 a_2 a_3 \bar{a}_4 \vee$
1	1	1	1	1	$a_1 a_2 a_3 a_4.$

DNF erhält man durch disjunktive Verknüpfung der Minterme, an denen $f = 1$ ist.

Beispiel: Erweiterung einer DF zur DNF (vgl. obige Beispiele)

$$\begin{aligned} f_{DF} &= a_1 a_2 \bar{a}_4 \underbrace{(a_3 \vee \bar{a}_3)}_1 \vee a_2 a_3 \bar{a}_4 \underbrace{(a_1 \vee \bar{a}_1)}_1 \vee a_3 a_4 \underbrace{(a_1 \vee \bar{a}_1)}_1 \underbrace{(a_2 \vee \bar{a}_2)}_1 \\ &= f_{DNF} \end{aligned}$$

2.1.10 Partiiell definierte Boolesche Funktionen

Bei nicht vollständig definierten Booleschen Funktionen, die meist in Form von Wertetabellen vorliegen, sind nur für bestimmte, nicht alle Belegungen der Eingangsvariablen, Funktionswerte definiert. Zur Bildung eines Booleschen Ausdrucks sind die Tabellen zu vervollständigen. Die Belegungen, die nicht in der gegebenen Tabelle enthalten sind, besitzen Funktionswerte, die für das Verhalten unerheblich sind (sogenannte "don't cares"), gekennzeichnet durch X. Diesen "don't cares" können nun beliebige Funktionswerte zugewiesen werden.

Beispiel 1 zu partiell definierten Boolesche Funktionen

a_1	a_2	a_3	x_1	x_2	
0	0	0	0	0	
0	0	1	0	0	
0	1	0	0	0	$x_1 = a_1 \bar{a}_2 \bar{a}_3 \vee a_1 a_2 \bar{a}_3$
0	1	1	0	0	$x_2 = a_1$
1	0	0	1	1	
1	0	1	X	1	
1	1	0	1	1	
1	1	1	X	1	

Beispiel 2 zu partiell definierten Booleschen Funktionen

a_1	a_2	a_2	f_1	f_2	
0	0	1	1		
0	1	0	0		$f_1 = \bar{a}_1 \bar{a}_2 a_3 \vee a_1 \bar{a}_2 \bar{a}_3$
1	0	0	1		
a_1	a_2	a_3			
0	0	0	X	1	$f_2 = \bar{a}_1 \bar{a}_2 \bar{a}_3 \vee$
0	0	1	1	1	$\bar{a}_1 \bar{a}_2 a_3 \vee$ } $\bar{a}_1 \bar{a}_2$
0	1	0	0	0	
0	1	1	X	0	
1	0	0	1	1	$a_1 \bar{a}_2 \bar{a}_3 \vee$ } $a_1 \bar{a}_2$ } a_1
1	0	1	X	1	$a_1 \bar{a}_2 a_3 \vee$ } $a_1 a_2$ }
1	1	0	X	1	$a_1 a_2 \bar{a}_3 \vee$ }
1	1	1	X	1	$a_1 a_2 a_3$

$$= \bar{a}_1 \bar{a}_2 \vee a_1$$

f_1 ist die DNF für die Boolesche Funktion, bei der durch Auswahl der Minterme, für die $f = 1$ gibt, implizit bei allen anderen Belegungen $f = 0$ angenommen ist.

Eine geschickte Nutzung der "don't cares" liefert eine einfachere Funktion, jedoch ist diese intuitive Vorgehensweise unbefriedigend. Erforderlich ist eine systematische Vorgehensweise.

2.2 Logische Schaltungen, Schaltnetz und Schaltwerk

Allgemein hat eine logische Schaltung die Aufgabe, binäre Eingabewerte $\{e_1, e_2, \dots, e_n\}$ in binäre Ausgabewerte $\{a_1, a_2, \dots, a_m\}$ zu transformieren.



Bild 2.2: Logische Schaltung

2.2.1 Schaltwerke

Schaltwerke realisieren sequentielle Boolesche Algorithmen. Es hängen hierbei die Ausgangssignale nicht nur von den momentanen Eingangssignalen, sondern von früheren Eingangssignalen und auch Ausgangssignalen ab. Die technische Realisierung bedarf daher Speicherelemente und ggf. Rückführungen von Ausgangsvariablen auf den Eingang (siehe Kapitel 4).

2.2.2 Schaltnetze

Schaltnetze sind technische Realisierungen Boolescher Funktionen. Es liegt eine logische Kombinationsschaltung vor, die die Booleschen Variablen $\{e_1, e_2, \dots, e_n\}$ verknüpft, so daß die Ausgangsvariablen $\{a_1, a_2, \dots, a_m\}$ verzögerungsfrei hieraus hervorgehen. Schaltnetze lassen sich allgemein durch ein Bündel Boolescher Funktionen beschreiben. Eingangs- und Ausgangsvariablen können auch als Vektoren aufgefaßt werden:

$$\begin{aligned} \underline{a} &= f(\underline{e}) \quad \text{mit} \\ a_1 &= f_1(e_1, e_2, \dots, e_n) \\ a_2 &= f_2(e_1, e_2, \dots, e_n) \\ &\vdots \\ a_m &= f_m(e_1, e_2, \dots, e_n) \end{aligned}$$

Die einzelnen Funktionen lassen sich getrennt behandeln. Zweckmäßig ist häufig allerdings, gleiche Teilausdrücke in den Funktionen nur einmal zu realisieren und gemeinsam zu verwenden.

2.2.3 Gatterschaltungen

Gatterschaltungen sind logische Verknüpfungsglieder mit einem oder mehreren Eingängen und nur einem Ausgang.

**Bild 2.3:** Gatterschaltung

Gatterschaltungen werden vielfach aus einigen wenigen Grundgattern gebildet. In der folgenden Tabelle sind die häufigsten Gattertypen in der gültigen DIN-Norm (DIN 40700), der noch häufig anzutreffenden älteren Darstellungsweise und der amerikanischen Norm angegeben.

Bezeichnung	Schaltsymbole nach DIN 40700 seit 1976 früher		amerik. Symbole	Funktion	Bemerkung
UND				$e_1 \wedge e_2 \dots \wedge e_n$	$e_1 \oplus e_2 \dots \oplus e_n$ (EXOR) (EXNOR)
ODER				$e_1 \vee e_2 \dots \vee e_n$	
Anti-valenz				$e_1 \nabla e_2 \nabla \dots \nabla e_n$	
Äqui-valenz				$e_1 \equiv e_2 \equiv \dots \equiv e_n$	
NOR				$\overline{e_1 \vee e_2 \dots \vee e_n}$	
NAND				$\overline{e_1 \wedge e_2 \dots \wedge e_n}$	
Nega-tion				$\bar{e}$	
log. Schlgt.				$\bar{e}_1 \wedge \bar{e}_2 \wedge e_3$	

Bild 2.4: Schaltsymbole für logische Schaltungen

2.2.4 Schaltungen mit nur einem Gattertyp

Alle booleschen Funktionen lassen sich mit den Operationen $\{\vee, \wedge, \neg\}$ darstellen. Zweckmäßig verwendet man auch andere Verknüpfungen. Es gibt in diesem Zusammenhang weitere Verknüpfungsmengen, die geeignet sind, alle Booleschen Funktionen darzustellen (vollständige Verknüpfungsmenge). Beispiele hierfür sind $\{\vee, \wedge, \neg\}$, $\{\neq, \wedge, \neg\}$ aber auch $\{\downarrow\}$ und $\{\mid\}$, d.h. NOR und NAND für sich alleine. Dies ist von großer technischer Bedeutung, da es offenbar gelingt, dann komplexe Gatterschaltungen aus nur einem einzigen Gattertyp aufzubauen, nämlich NAND- oder NOR-Gattern. Dies ist bedeutsam für die Reduktion der Gattertypvielfalt, aber

auch für einen gleichmäßigen IC-Entwurf. Zu beachten ist, daß für NAND und NOR nicht das Assoziativgesetz gilt.

2.3 Systematische Minimierungsverfahren für Schaltnetze

Die Minimierung von Schaltfunktionen ist eine Aufgabe deren Lösung für die Datenverarbeitungstechnik von großer Bedeutung ist. Bei der Realisierung einer Schaltfunktion durch Gatterbausteine wird versucht, einen möglichst minimalen Booleschen Ausdruck zu finden, der die Funktion beschreibt. Dies geschieht um eine kostengünstige Lösung zu finden, denn die Kosten einer solchen Schaltung sind abhängig von der Länge des zugrundeliegenden Ausdrucks. Schon in den Anfängen der Digitaltechnik hat sich gezeigt, dass die Vereinfachungsverfahren von Hand zu fehleranfällig und zeitaufwändig sind. Die Qualität der erreichten Lösungen hängt dabei stark von Intuition und Erfahrung ab. Von großer Bedeutung sind daher systematische Minimierungsverfahren.

Minimierungsverfahren erzeugen für einen Booleschen Ausdruck eine Form mit minimalem Schaltungsaufwand. Man unterscheidet

- (1) algebraische Verfahren
- (2) graphische Verfahren
- (3) tabellarische und algorithmische Verfahren

Erstere beruhen auf der Anwendung der Rechenregeln der logischen Algebra (besonders "de Morgan", "Distributivgesetze", "Absorptionsgesetz"). Häufig kann man wesentlich vereinfachen durch disjunktive oder konjunktive Ergänzungen bereits vorhandener Terme, denn: $a = a \wedge a \wedge a \wedge \dots \wedge a$, $a = a \vee a \vee a \vee \dots \vee a$ sowie nachfolgende Anwendung der Distributiv- bzw. Absorptionsgesetze.

$$y = \bar{a}\bar{b}c \vee \bar{a}b\bar{c} \vee \bar{a}bc \vee a\bar{b}\bar{c}$$

Beispiel:

$$= (\bar{a}\bar{b}c \vee \bar{a}bc) \vee (\bar{a}b\bar{c} \vee a\bar{b}\bar{c})$$

$$= (\bar{a}c) \vee (\bar{a}b) \vee (a\bar{b})$$

Algebraische Verfahren beruhen stark auf Intuition und Erfahrung. Die anderen Verfahren lassen sich leichter systematisieren. Graphische Verfahren lassen sich von Hand ausführen, liefern eine schnelle Übersicht (und Verständnis) für die Minimierung bis zu sechs Variablen. Tabellarische bzw. algorithmische Verfahren sind wesentlich mächtiger und für eine Softwarerealisierung geeignet. Graphische Verfahren werden im folgenden ausführlich behandelt, da sie auch ein Grundverständnis der allgemeinen Zusammenhänge fördern. Es folgt dann die

Darstellung eines tabellarischen Verfahrens (Quine/McCluskey), das für eine Softwarerealisierung geeignet ist.

2.3.1 Karnaugh-Veitch (KV)-Diagramme

Die hier behandelte Methode zur systematischen Vereinfachung von Schaltfunktionen basiert auf den Arbeiten von Karnaugh und Veitch, die Anfang der fünfziger Jahre veröffentlicht wurden. Es handelt sich dabei um ein graphisches Minimierungsverfahren, das es ermöglicht sehr schnell die Minimalform von Booleschen Ausdrücken zu bestimmen. Aus diesem Grunde ist seine Anwendung heute weit verbreitet. Ein Nachteil des Verfahrens ist jedoch, daß es nur bei Funktionen von bis zu fünf Variablen effizient anzuwenden ist. Bei mehr Variablen wird das Verfahren schnell unhandlich und unübersichtlich. Der Vorteil, dass es vor allem schnell zu einer Lösung führt, geht dann verloren.

Grundlage aller graphischen Minimierungsverfahren ist eine spezielle graphische Darstellung von Booleschen Funktionen. Wir beschränken uns hier auf die von Karnaugh und Veitch vorgeschlagene Darstellung, die sogenannten Karnaugh-Veitch-Diagramme, kurz KV-Diagramme genannt.

KV-Diagramme sind graphische Darstellungen für Boolesche Funktionen. Für eine n -stellige Boolesche Funktion (n Variablen) gibt es 2^n Kombinationen der Eingangsvariablen (Belegungen) und 2^n Funktionswerte. Ein KV-Diagramm besteht aus 2^n quadratischen Einzelfeldern, die zu einem Rechteckschema angeordnet sind. Jedem Feld ist eine Belegung der Variablen mit 0,1-Werten zugeordnet. Jedes Feld repräsentiert einen Minterm ($=1$) bzw. einen Maxterm ($=0$):

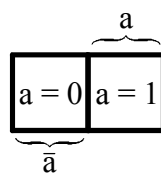


Bild 2.5: KV-Diagramm für eine Variable

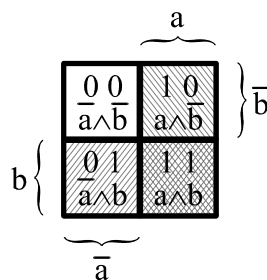


Bild 2.6: KV-Diagramm für zwei Variable

Für jede weitere Variable ist das Diagramm zu verdoppeln, der alte Bereich gilt dann für die negierte, der neue Bereich für die nichtnegierte neue Variable. Hiermit lassen sich im Prinzip alle möglichen Booleschen Funktionen darstellen.

KV-Diagramme gestatten neben der Darstellung der Funktion zusätzlich die Minimierung einer logischen Funktion durch Zusammenfassen benachbarter Felder. Voraussetzung ist, daß zwei aneinandergrenzende Felder sich genau in einer Variablen unterscheiden. Eine mögliche Anordnung bei drei Variablen zeigt das Bild.

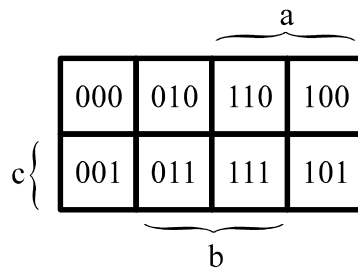


Bild 2.7: KV-Diagramm für drei Variable (z. B. 011: a=0, b=1, c=1)

Die Belegung ist als Dualzahl angegeben. Felder, die sich nur in einer Variablen unterscheiden, liegen in gleichen Zeilen oder Spalten. Felder werden als "benachbart" bezeichnet, wenn sie sich nur in einer Variablen unterscheiden. Auch Randfelder sind benachbart, die sich in einer Zeile oder Spalte gegenüber liegen.

Der Begriff Nachbarschaft von Randfeldern kann durch eine Zylinderdarstellung veranschaulicht werden.

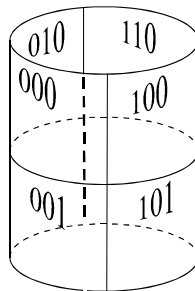


Bild 2.8: KV-Diagramm in Zylinderdarstellung

Ersichtlich können durch geeignetes Aufschneiden des Zylinders verschiedenen Anordnungen des KV-Diagramms erzeugt werden.

Eine der grundlegenden Möglichkeiten zur algebraischen Vereinfachung besteht darin, Terme, die sich nur in einer Variablen unterscheiden (negiert bzw. nicht negiert), zusammen zu fassen. z.B.

$$abc \vee ab\bar{c} = ab \left(\underbrace{c \vee \bar{c}}_1 \right) = ab$$

Diese Zielsetzung begründet den eingeführten Nachbarschaftsbegriff.

Das Entwicklungsschema für mehrere Variable zeigt das folgende Bild. Für jede hinzukommende Variable wird das bereits vorhandene Diagramm an der unteren bzw. rechten Seite gespiegelt. Zu beachten ist, daß ab 5 Variablen sich beim Übergang von einem Feld zum anderen sich nur eine Variable von vieren ändern kann.

Bild 2.9 zeigt das Entwicklungsschema für Boolesche Ausdrücke bis zu 6 Variablen.

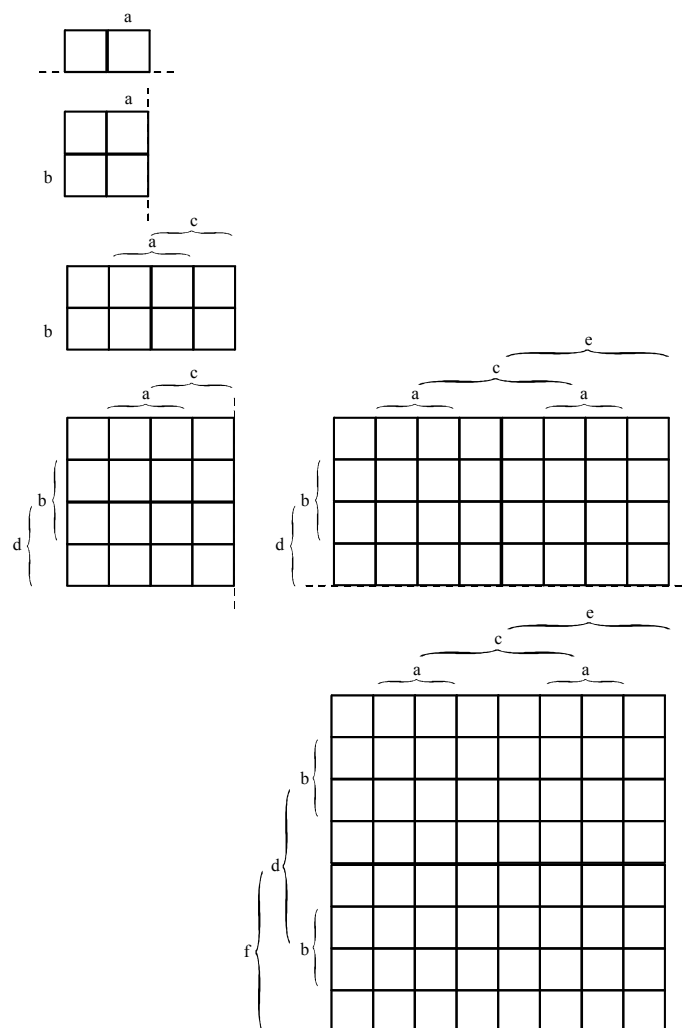


Bild 2.9: Entwicklungsschema für KV-Diagramme

Für die 5. Variable ergibt sich kein zusammenhängender Bereich mehr. "Benachbart" sind bei geteilten Bereichen auch Felder, die weder aneinander grenzen noch am Rand liegen.

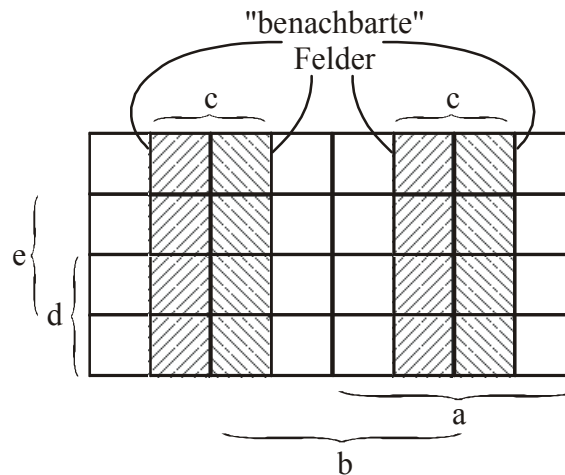


Bild 2.10: KV-Diagramm für 5 Variable

Die im Bild gleichartig schraffierten Bereiche unterscheiden sich nur in der Variablen a. Sie sind in diesem Sinne benachbart. Die Wahl der Zuordnung der Variablen ist im übrigen willkürlich. Zweckmäßig ist allerdings ein stets gleiches Belegungsschema.

Die erwähnte „Nachbarschaft“ der auseinanderfallenden Bereiche kann durch eine räumliche (Würfel-) Darstellung plausibel gemacht werden, bei der die beiden Hälften aufeinander geklappt sind.

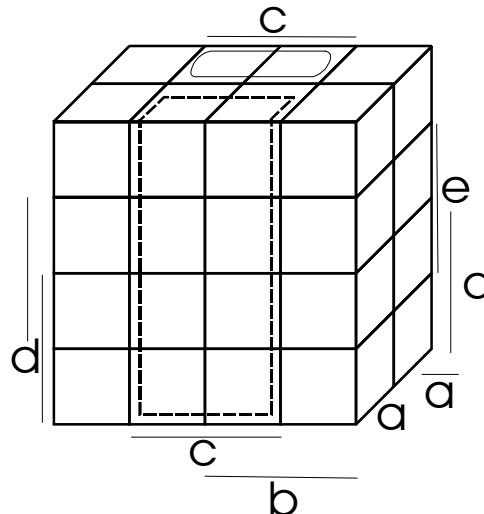


Bild 2.11: Würfel-Darstellung für ein KV-Diagramm mit fünf Variablen

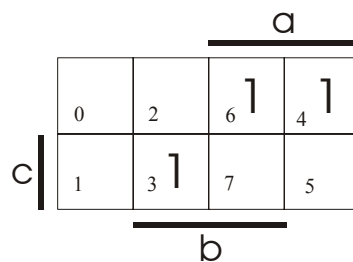
2.3.1.1 Darstellung Boolescher Funktionen in KV-Diagrammen

Bei Vorliegen einer Wertetabelle für die Boolesche Funktion erfolgt die KV-Diagramm-Darstellung sehr einfach durch Übertragen der Funktionswerte ins KV-Diagramm (incl. don't care-Felder). Zweckmäßig und übersichtlich ist eine dezimale Beschriftung entsprechend dem folgenden Bild, möglich sind dabei Eins- und Null-Muster.

Wertetabelle	Dezimale Feldbeschriftung	Eins - Muster	Null - Muster																																																																																													
<table><tr><td>n</td><td>a</td><td>b</td><td>c</td><td>f</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>2</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>3</td><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>4</td><td>1</td><td>0</td><td>0</td><td>X</td></tr><tr><td>5</td><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>6</td><td>1</td><td>1</td><td>0</td><td>0</td></tr><tr><td>7</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	n	a	b	c	f	0	0	0	0	0	1	0	0	1	1	2	0	1	0	1	3	0	1	1	0	4	1	0	0	X	5	1	0	1	0	6	1	1	0	0	7	1	1	1	1	<table><tr><td colspan="4">a</td></tr><tr><td>0</td><td>2</td><td>6</td><td>4</td></tr><tr><td>1</td><td>3</td><td>7</td><td>5</td></tr><tr><td colspan="4">b</td></tr></table> c	a				0	2	6	4	1	3	7	5	b				<table><tr><td colspan="4">a</td></tr><tr><td></td><td>1</td><td></td><td>X</td></tr><tr><td>1</td><td></td><td>1</td><td></td></tr><tr><td colspan="4">b</td></tr></table> c	a					1		X	1		1		b				<table><tr><td colspan="4">a</td></tr><tr><td>0</td><td></td><td>0</td><td>X</td></tr><tr><td></td><td>0</td><td></td><td>0</td></tr><tr><td colspan="4">b</td></tr></table> c	a				0		0	X		0		0	b			
n	a	b	c	f																																																																																												
0	0	0	0	0																																																																																												
1	0	0	1	1																																																																																												
2	0	1	0	1																																																																																												
3	0	1	1	0																																																																																												
4	1	0	0	X																																																																																												
5	1	0	1	0																																																																																												
6	1	1	0	0																																																																																												
7	1	1	1	1																																																																																												
a																																																																																																
0	2	6	4																																																																																													
1	3	7	5																																																																																													
b																																																																																																
a																																																																																																
	1		X																																																																																													
1		1																																																																																														
b																																																																																																
a																																																																																																
0		0	X																																																																																													
	0		0																																																																																													
b																																																																																																

Bild 2.12: Wahrheitstabelle und KV-Diagramm

Wenn die Funktion durch einen Booleschen Ausdruck gegeben ist, könnte man daraus die vollständige Wertetabelle bestimmen und die Funktionswerte wie oben beschrieben ins Diagramm eintragen. Diese Lösung ist allerdings aufwendig und kann vermieden werden, wenn man den gegebenen Ausdruck direkt in das Diagramm übertragen kann. Dies ist möglich und soll im folgenden beschrieben werden. Wir gehen zunächst nur von Ausdrücken aus, die in einer *disjunktiven* Form gegeben sind. Der einfachste Fall liegt vor, wenn eine disjunktive Normalform gegeben ist. Da jedem Minterm genau ein Feld zugeordnet ist, müssen wir nur diejenigen Felder suchen und kennzeichnen, die den Mintermen der DNF entsprechen.

Bild 2.13: Beispiel: $f(a,b,c) = abc \vee ab\bar{c} \vee \bar{a}bc$

Die Darstellung von der **disjunktiven Normalform** ausgehend ergibt sich im KV-Diagramm unmittelbar, da jedem Minterm genau ein Feld zugeordnet ist. Ist nur eine DF gegeben, so kennzeichnet ein Term die Konjunktion aus einigen (nicht allen) Variablen. Nur die sich überdeckenden Flächen dieser Variablen erhalten für diesen Term Einsen.

In anderen disjunktiven Formen sind Einzelterme enthalten, die aus Konjunktionen von weniger Variablen bestehen. Bei solchen Termen ist die Anzahl der zu kennzeichnenden Felder größer. Um für eine disjunktive Form die Eintragungen ins KV-Diagramm zu finden, betrachten wir die einzelnen Terme nacheinander.

Um das Verfahren zu verdeutlichen, zeichnen wir zunächst für jeden Term ein eigenes Diagramm, in dem wir die Bereiche jeder einzelnen darin enthaltenen Variablen unterschiedlich schraffieren. Da wir es mit einer Konjunktion der Variablen zu tun haben, dürfen wir nur die Felder kennzeichnen, die allen Variablen gemeinsam sind. Dies sind die Felder, die von allen Schraffuren überdeckt werden. Diagramme für die einzelnen Terme werden zu einem resultierenden KV-Diagramm zusammengefügt, indem man in jedes Feld eine „1“ einträgt, das in mindestens einem Einzeldiagramm eine „1“ enthält; denn die Einzelterme sind ja disjunktiv miteinander verknüpft.

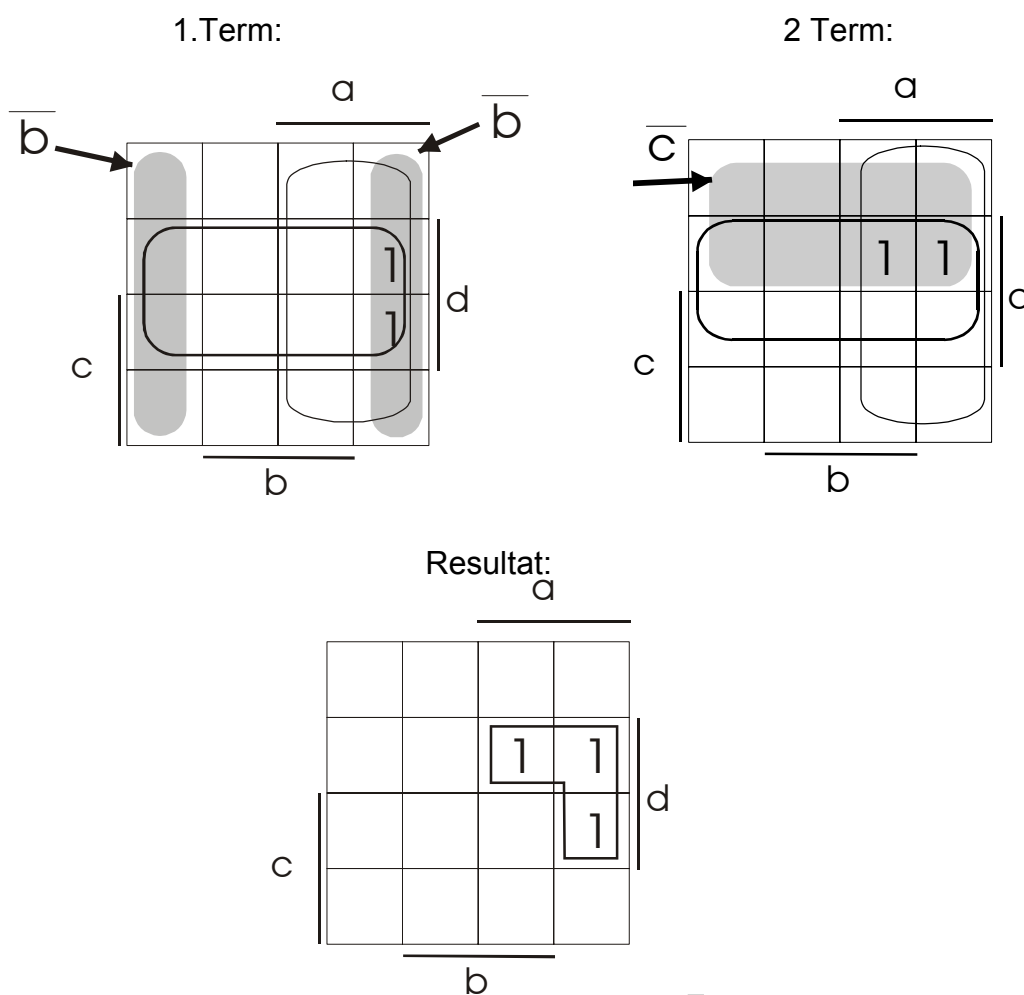


Bild 2.14: Beispiel : $f(a,b,c,d) = a\bar{b}d \vee a\bar{c}d$

Wenn man das Prinzip für diese Eintragungen erkannt hat, kommt man natürlich ohne Schraffuren aus und kann die Markierungen für alle Terme sofort in ein einziges Diagramm eintragen.

Ist eine konjunktive Form gegeben, so sind die Flächen der (konjunktiv verknüpften) Disjunktionen zu ermitteln und dann deren Schnittflächen mit Einsen zu belegen. Einfacher bildet man zur gegebenen konjunktiven Form den dualen Ausdruck und

trägt in die entsprechenden Felder Nullen ein, denn bei einer vollständig definierten Booleschen Funktion sind 0- und 1- Muster zueinander komplementär.

Beispiel: Konjunktive Form einer Funktion:

Wenn eine Funktion in Konjunktiver Form gegeben ist, ist das Erstellen eines KV-Diagramm deutlich aufwendiger als bei disjunktiver Form. Dazu müsste im Prinzip jeder disjunktive Einzelterm in ein eigenes KV-Diagramm eingetragen werden. Da diese Terme jedoch konjunktiv verknüpft sind, ist das resultierende KV-Diagramm nur in denjenigen Feldern mit einer „1“ besetzt, die in *allen* Einzeldiagrammen auch eins sind.

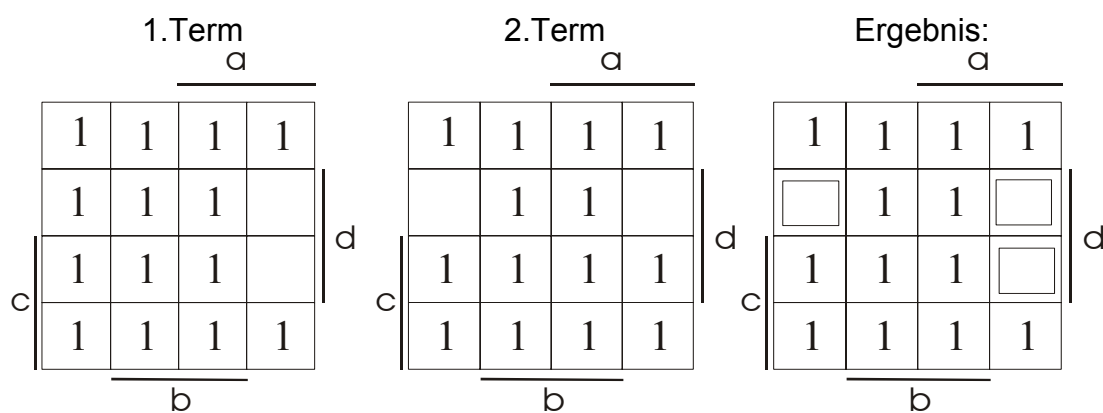


Bild 2.15: Beispiel : $f(a,b,c,d) = (\bar{a} \vee b \vee \bar{d}) \wedge (b \vee c \vee \bar{d})$, direkte Lösung

Man bildet deshalb meistens den dualen Ausdruck zur gegebenen konjunktiven Form und trägt dann in die entsprechenden Felder „0“ ein:

$$f(a,b,c,d) = (\bar{a} \vee b \vee \bar{d}) \wedge (b \vee c \vee \bar{d})$$

$$\bar{f} = (\overline{\bar{a} \vee b \vee \bar{d}}) \vee (\overline{b \vee c \vee \bar{d}})$$

$$= a\bar{b}d \vee \bar{b}\bar{c}d$$

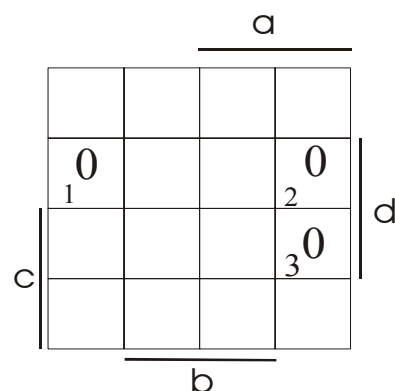
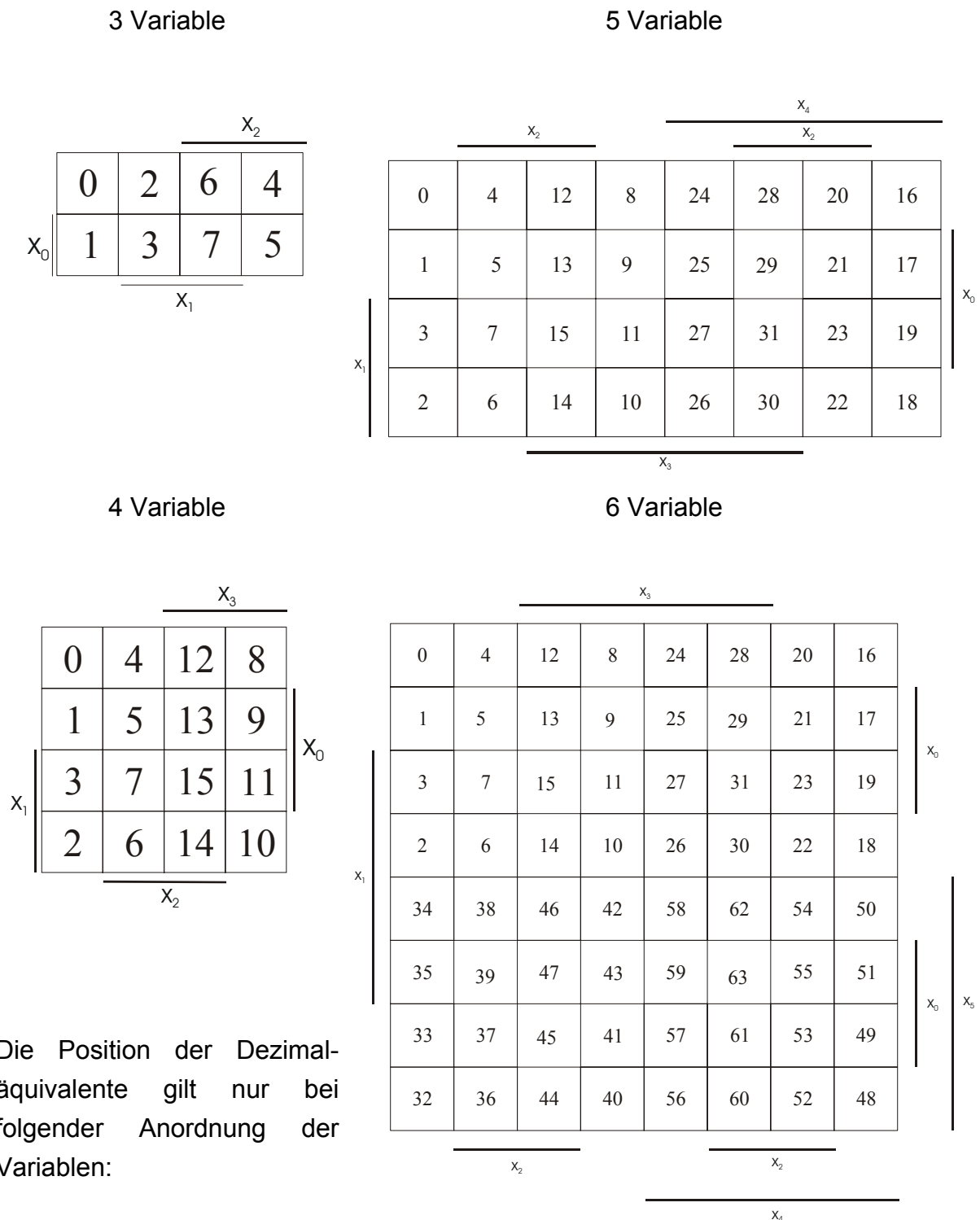


Bild 2.16: Beispiel : $f(a,b,c,d) = (\bar{a} \vee b \vee \bar{d}) \wedge (b \vee c \vee \bar{d})$, Lösung über dualen Ausdruck

Zweckmäßige Variablenverteilung im KV-Diagramm:



Die Position der Dezimal-äquivalente gilt nur bei folgender Anordnung der Variablen:

abcd, wobei a das MSB ist

oder $x_3x_2x_1x_0$

Bild 2.17: Zweckmäßige Variablenverteilung im KV-Diagramm

Diese Belegung der Variablen wird in Vorlesung, Übungen und Klausuren einheitlich verwendet.

2.3.1.2 Minimierung mit KV-Diagrammen

Die Minimierung kann so vorgenommen werden, daß Gruppen benachbarter Felder (s. Definition), die gleich gekennzeichnet sind, zu einem Block vereint werden und dann eine Variable fortgelassen wird. Ist beispielsweise

$$f = (\bar{a} \wedge b \wedge \bar{c} \wedge \bar{d}) \vee (\bar{a} \wedge \bar{b} \wedge \bar{c} \wedge d) \vee (a \wedge \bar{b} \wedge \bar{c} \wedge d) \vee (\bar{a} \wedge b \wedge c \wedge \bar{d})$$

so lassen sich der erste und letzte Term sowie die mittleren Terme algebraisch vereinfachen:

$$\begin{aligned} f &= (\bar{a} \wedge b \wedge \bar{d}) \wedge \underbrace{(\bar{c} \vee c)}_1 \vee (\bar{b} \wedge \bar{c} \wedge d) \wedge \underbrace{(\bar{a} \vee a)}_1 \\ &= \bar{a} \wedge b \wedge \bar{d} \vee \bar{b} \wedge \bar{c} \wedge d \end{aligned}$$

Das KV-Diagramm im folgenden Bild enthält definitionsgemäß "benachbarte" Randfelder, die zusammengefaßt werden können.

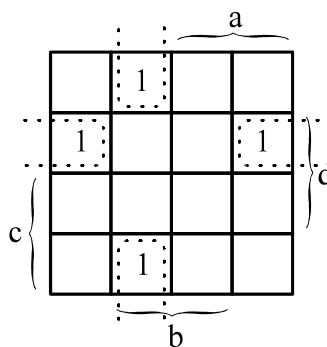
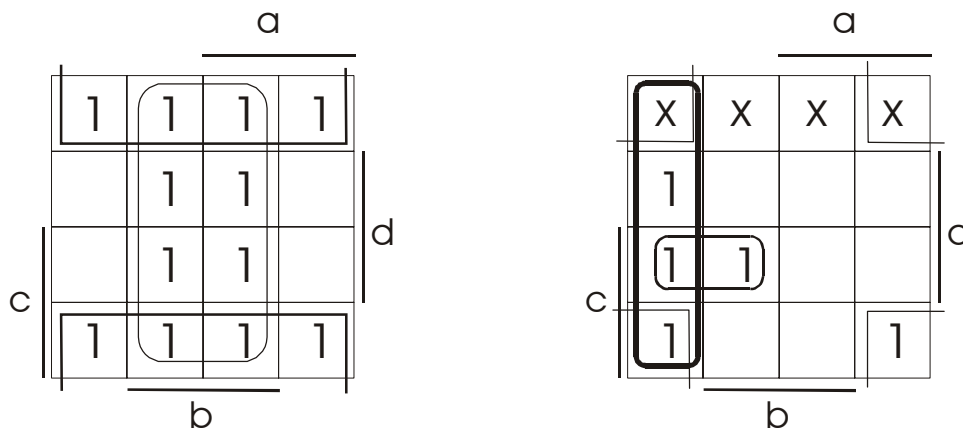


Bild 2.18: Minimierung mit Hilfe eines KV-Diagrammes

Eine Variable kann dann in einem Zweierblock (allgemein: n Variable in einem 2^n -Felder-Block) fortgelassen werden, und es wird er Block durch einen Term repräsentiert (vgl. algebraische Vereinfachung). Die Zusammenfassung von 1-Feldern führt stets aus disjunktive Formen für die Booleschen Ausdrücke, d.h. auf Schaltnetze in der Form von zweistufigen UND/ODER-Verknüpfungen.

Will man eine konjunktive Form realisieren, also ein UND/ODER-Schaltnetz aufbauen, so werden alle 0-Felder zu möglichst großen Blöcken zusammengefaßt (wie zuvor die 1-Felder). Daraus wird die disjunktive Minimalform abgelesen. Der gewonnene Ausdruck beschreibt das Komplement $\bar{f}$, d.h. den dualen Ausdruck. Durch Komplementierung (Gesetz von de Morgan !) erhält man die gesuchte konjunktive Form.

„Don't care“-Terme sind frei wählbar. Sie werden in jedem Falle benutzt, um möglichst große 1- bzw. 0-Blöcke zu bilden.



$$f(a, b, c, d) = b \vee \bar{d}$$

$$f(a, b, c, d) = \bar{a}\bar{b} \vee \bar{a}cd \vee b\bar{d}$$

Bild 2.19: Beispiel zur Verwendung von „Don't care“-Plätzen

Allgemein ist bei der Zusammenfassung von Feldern zu beachten, daß nur die Zusammenfassung zu den größtmöglichen Blöcken eine minimale Form ergeben kann. In einigen Fällen ist es möglich gleich große Blöcke mit verschiedenen Feldern zu bilden. In einem solchen Fall entstehen äquivalente minimale Formen (mit gleicher Länge).

2.3.2 Minimierungsverfahren nach Quine-McCluskey

Dieses Minimierungsverfahren ist ein tabellarisches Verfahren für die Vereinfachung von Schaltfunktionen mit vielen (>4) Variablen. Es kann algorithmisch abgewickelt werden und ist damit zum Rechnereinsatz geeignet. Das Verfahren basiert wie die übrigen auf der Vereinfachung von Termen, die sich nur in einer Variable unterscheiden. In diesem Fall wird jedoch die Suche nach solchen Termen systematisch betrieben. Am Ende des Suchverfahrens sind die "Primterme" der Funktion ermittelt. Ein "Primterm" einer Funktion ist eine Konjunktion (UND-Verknüpfung), aus der sich keine Variable streichen läßt, ohne den Wert der Funktion zu ändern. Im KV-Diagramm sind Primterme anschaulich Blöcke mit einer maximalen Zahl von Einsen. Im dargestellten KV-Diagramm sind die Blöcke 1, 2 und 3 Primterme. Ein Primterm wird als "wesentlicher Primterm" bezeichnet, falls er eine 1 enthält, die in keinem anderen Primterm enthalten ist. Dies trifft hier nur für die Blöcke 1 und 3 zu.

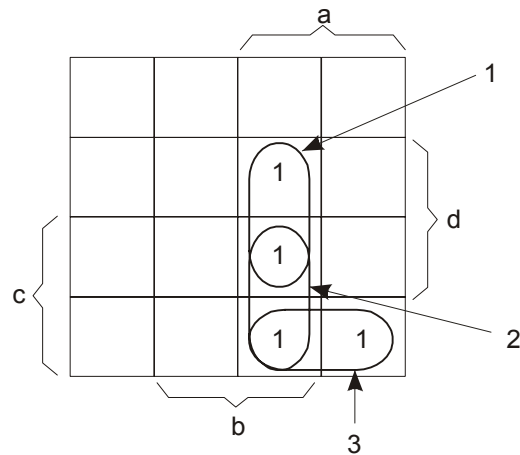


Bild 2.20: Wesentliche Primterme im KV-Diagramm

Zu Beginn des Verfahrens wird die Wertetabelle der zu vereinfachenden Funktion aufgestellt. In unserem Beispiel ist es die Funktion y , die von Variablen a , b , c und d abhängt:

Nr.	a	b	c	d	y
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	0

In einer weiteren Tabelle werden alle Minterme eingetragen, für die die Funktion erfüllt ist, d.h. den Wert 1 annimmt. Die Minterme werden nach der Anzahl der Einsen gruppiert.

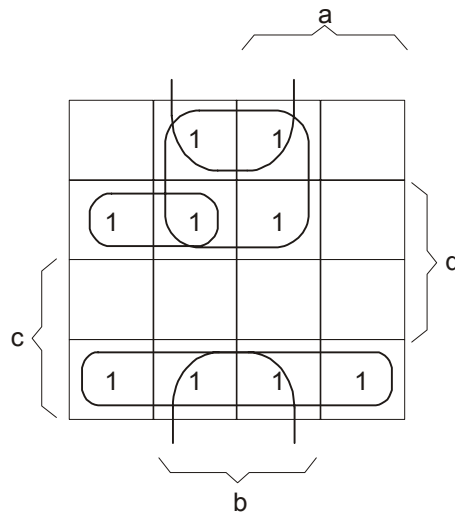
Nr.	a b c d	zusammengefasste Terme 1. Ordnung	a b c d	zusammengefasste Terme 2. Ordnung	a b c d
1	0 0 0 1 ✓	1,5	0 - 0 1 A	2,6;10,14	- - 1 0 B
2	0 0 1 0 ✓	2,6	0 - 1 0 ✓	2,10;6,14	- - 1 0 (doppelt)
4	0 1 0 0 ✓	2,10	- 0 1 0 ✓	4,5;12,13	- 1 0 - C
5	0 1 0 1 ✓	4,5	0 1 0 - ✓	4,6;12,14	- 1 - 0 D
6	0 1 1 0 ✓	4,6	0 1 - 0 ✓	4,12;5,13	- 1 0 - (doppelt)
10	1 0 1 0 ✓	4,12	- 1 0 0 ✓	4,12;6,14	- 1 - 0 (doppelt)
12	1 1 0 0 ✓	5,13	- 1 0 1 ✓		
13	1 1 0 1 ✓	6,14	- 1 1 0 ✓		
14	1 1 1 0 ✓	10,14	1 - 1 0 ✓		
		12,13	1 1 0 - ✓		
		12,14	1 1 - 0 ✓		

Jeder Term einer Gruppe wird nun mit den Termen der benachbarten Gruppen verglichen. Ausdrücke, die sich jeweils nur in einer Variablen unterscheiden, werden durch Streichen der unterschiedlichen Stelle vereinfacht. Die zusammengefassten Terme werden mit einem Haken gekennzeichnet. Die nicht abgehakten Terme lassen sich nicht weiter vereinfachen und stellen somit Primterme dar.

Die Terme werden nun erneut zu Gruppen zusammengefaßt und nach dem gleichen Verfahren weiter vereinfacht. Gleichzeitig wird mit in die Tabelle aufgenommen, aus welchen Ausdrücken die Terme hervorgegangen sind. Durch die Vereinfachungen kann es vorkommen, daß verschiedene Zusammenfassungen auf gleiche Terme führen, von denen nur jeweils einer berücksichtigt werden muß. Die mit A, B, C und D gekennzeichneten Ausdrücke stellen die Primterme dar. Die disjunktive Verknüpfung aller Primterme ergibt

$$y = \bar{a}\bar{c}d \vee c\bar{d} \vee b\bar{c} \vee b\bar{d}.$$

In der minimalen disjunktiven Form sind nicht notwendigerweise alle Primterme enthalten. Im KV-Diagramm erkennt man, daß der Term $b \wedge \bar{d}$ nicht "wesentlich" ist.



Minterme	✓	✓	✓	✓	✓	✓	✓	✓	✓
Primterme	1	2	4	5	6	10	12	13	14
A: 1, 5	*			*					
B: 2, 6; 10, 14		*			*	*			*
C: 4, 5; 12, 13			*	*			*	*	
D: 4, 6; 12, 14			*		*		*		*

Bild 2.21: Beispiel: Bestimmung wesentlicher Primterme

Das Quine-McCluskey-Verfahren verwendet zur Bestimmung der wesentlichen Primterme eine Tabelle, in der die Minterme in den Spalten und die Primterme in den Zeilen eingetragen sind. Innerhalb der Tabelle sind die Minterme, die in einem Primterm zusammengefaßt sind, mit einem Stern gekennzeichnet. In der ersten Zeile für Primterm A werden z.B. die Spalten 1 und 5 mit einem Stern markiert.

Dann werden die Spalten gewählt, die nur einen Stern enthalten. Im betrachteten Beispiel sind dies die Spalten 1, 2, 10 und 13. Damit sind die wesentlichen Primterme A, B und C festgelegt. Die Minterme, die mit den Primtermen A, B und C erfaßt sind, können abgehakt werden. Jetzt muß überprüft werden, ob Spalten in der Tabelle enthalten sind, die noch nicht mit den Primtermen A, B und C erfaßt wurden. Dies ist nicht der Fall, und man erhält als Lösung

$$y = \bar{a}\bar{c}d \vee c\bar{d} \vee b\bar{c}.$$

Falls irgendwelche Terme noch nicht erfaßt sind, müssen weitere Zeilen ausgewählt werden, bis alle Terme abgedeckt sind. Ziel ist es dann, alle Terme mit einer minimalen Zahl von Zeilen zu erfassen, und dabei, falls möglich, die Primterme mit den wenigsten Variablen zu verwenden.

2.4 Ausgewählte Beispiele zu logischen Schaltungen

In diesem Abschnitt werden beispielhaft einige wichtige logische Schaltnetze behandelt: Dualzahlenvergleicher, 1- aus - N - Decoder, Prioritäts-Encoder, Multiplexer, und Demultiplexer.

2.4.1 Zweistelliger Dualzahlenvergleicher

Eine natürliche Zahl N_2 , im Dualzahlensystem dargestellt durch ihre Ziffern (Bits) $Z_{K-1}, \dots, Z_1, Z_0$ hat einen Wert entsprechend folgender Gleichung:

$$N_2 = \sum_{k=0}^{K-1} Z_k \cdot 2^k = Z_{K-1} \cdot 2^{K-1} + \dots + Z_1 \cdot 2^1 + Z_0 \cdot 2^0$$

Ein Dualzahlenvergleicher bildet die logische Aussage für 2 natürliche Dualzahlen N_{2a}, N_{2b}

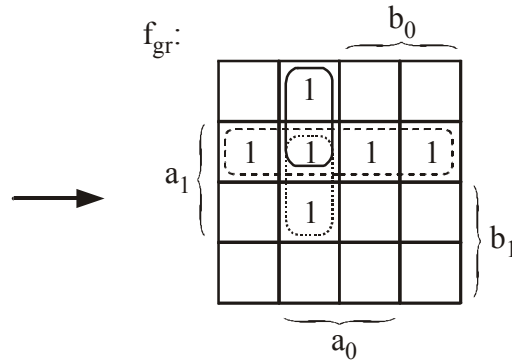
$$\begin{aligned} &> & f_{gr} &= (N_{2a} > N_{2b}) \\ N_{2a} &< N_{2b} & \text{ bzw. } & f_{kl} = (N_{2a} < N_{2b}) \\ &= & f_{gl} &= (N_{2a} = N_{2b}) \end{aligned}$$

d.h. das Schaltnetz hierfür enthält K Eingänge und 3 Ausgänge. Der Entwurf erfolgt wie dargestellt über Wahrheitstabelle, Karnaugh-Diagramm, Minimierung und Bildung der logischen Funktionen. Für zweistellige Dualzahlen ist als Beispiel diese Vorgehensweise dargestellt.

WahrheitstabelleKarnaugh-Veitch-Diagramm

(Minimierbare Felder sind umrandet)

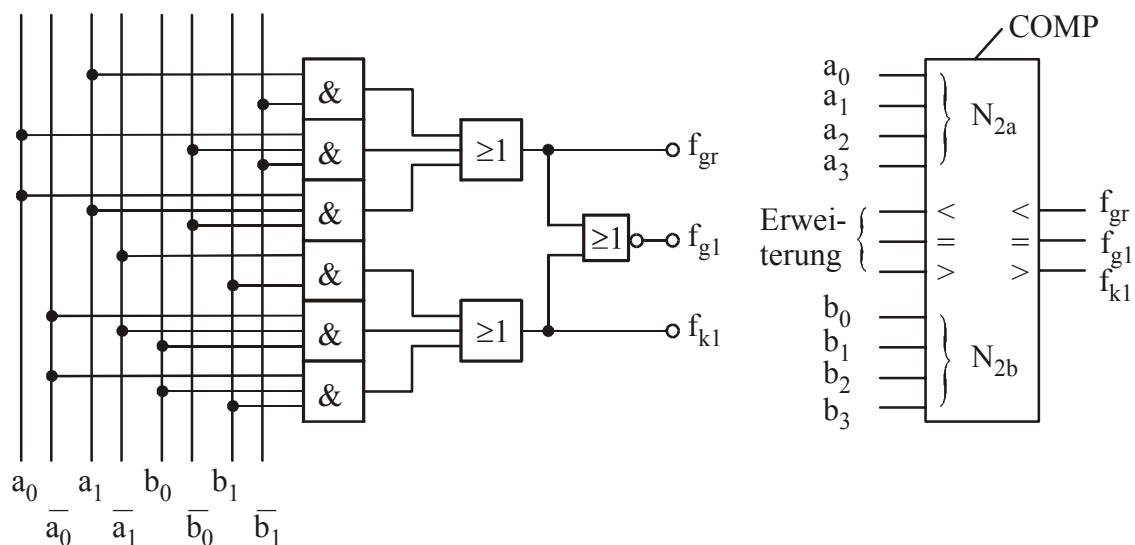
a_1	a_0	b_1	b_0	f_{gr}	f_{kl}	f_{gl}
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	0	1

**Bild 2.22:** Dualzahlenvergleich, KV-DiagrammLogische Funktionen:

$$f_{gr} = (a_1 \wedge \bar{b}_1) \vee (a_0 \wedge \bar{b}_1 \wedge \bar{b}_0) \vee (a_0 \wedge a_1 \wedge \bar{b}_0)$$

$$f_{kl} = (\bar{a}_1 \wedge b_1) \vee (b_0 \wedge \bar{a}_1 \wedge \bar{a}_0) \vee (b_0 \wedge b_1 \wedge \bar{a}_0)$$

$$f_{gl} = \bar{f}_{gr} \wedge \bar{f}_{kl} = \overline{f_{gr} \vee f_{kl}}$$

Resultierendes Schaltnetz:**Bild 2.23:** Dualzahlenvergleich

2.4.2 1-aus-N-Decoder, Prioritäts-Encoder

Ein Code-Umsetzer ordnet den Zeichen eines Codes A die Zeichen eines Codes B zu. Ein *1-aus-N-Decoder* hat N Ausgänge und Id N Eingänge. Ein Ausgang n liefert dann logisch 1, wenn die eingegebene Dualzahl gleich der Nummer n ist. Das spezifische Schaltnetz wird am Beispiel eines 1 aus 8 Decoders dargestellt.

Wahrheitstabelle

a_2	a_1	a_0	f_7	f_6	f_5	f_4	f_3	f_2	f_1	f_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

Karnaugh-Veitch-Diagramm

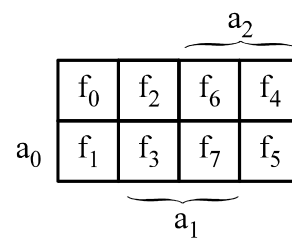


Bild 2.24: 1-aus-N-Decoder, Wahrheitstabelle und KV-Diagramm

Die Booleschen Ausgangsfunktionen erhält man durch Bildung der zugehörigen Minterme, z. B. $f_6 = a_2 \wedge a_1 \wedge \bar{a}_0$, zugehöriges Schaltnetz:

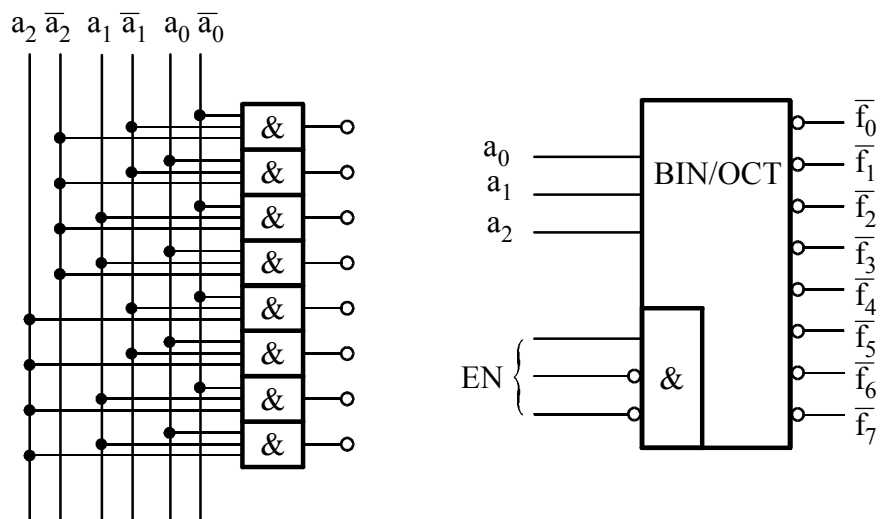


Bild 2.25: 1-aus-N-Decoder

Ein *Prioritäts-Encoder* bildet die Umkehrung des 1-aus-N-Decoders. An seinen Ausgängen tritt diejenige Dualzahl auf, die der höchsten Eingangszahl entspricht, an der eine 1 anliegt.

Wahrheitstabelle eines 1 aus 8 Prioritäts-Encoders:

Dez	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0	a_2	a_1	a_0
(0)–1	0	0	0	0	0	0	0	1	0	0	0
2–3	0	0	0	0	0	0	1	x	0	0	1
4–7	0	0	0	0	0	1	x	x	0	1	0
8–15	0	0	0	0	1	x	x	x	0	1	1
16–31	0	0	0	1	x	x	x	x	1	0	0
32–63	0	0	1	x	x	x	x	x	1	0	1
64–127	0	1	x	x	x	x	x	x	1	1	0
128–255	1	x	x	x	x	x	x	x	1	1	1

Als Beispiel sei ein einfacher 1 aus 4 Prioritäts-Encoder betrachtet:

Wahrheitstabelle:

Dez	e_3	e_2	e_1	e_0	a_1	a_0
(0)–1	0	0	0	1	0	0
2–3	0	0	1	x	0	1
4–7	0	1	x	x	1	0
8–15	1	x	x	x	1	1

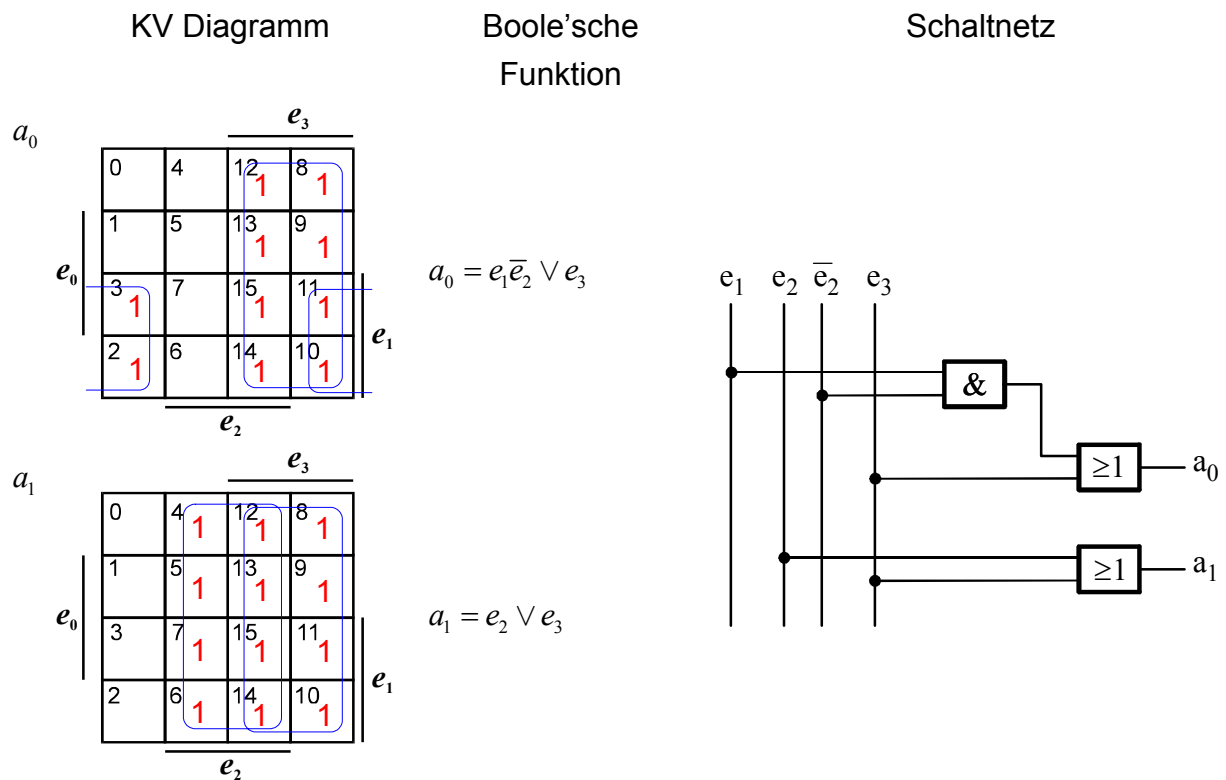


Bild 2.26: Prioritäts-Encoder

2.4.3 Multiplexer, Demultiplexer

Mit einem *Demultiplexer* kann man Eingangsdaten an verschiedene auswählbare Ausgänge verteilen. Der über einen 1-aus-N-Decoder adressierte Ausgang nimmt den Wert der Eingangsvariablen an (vgl. Schaltnetz).

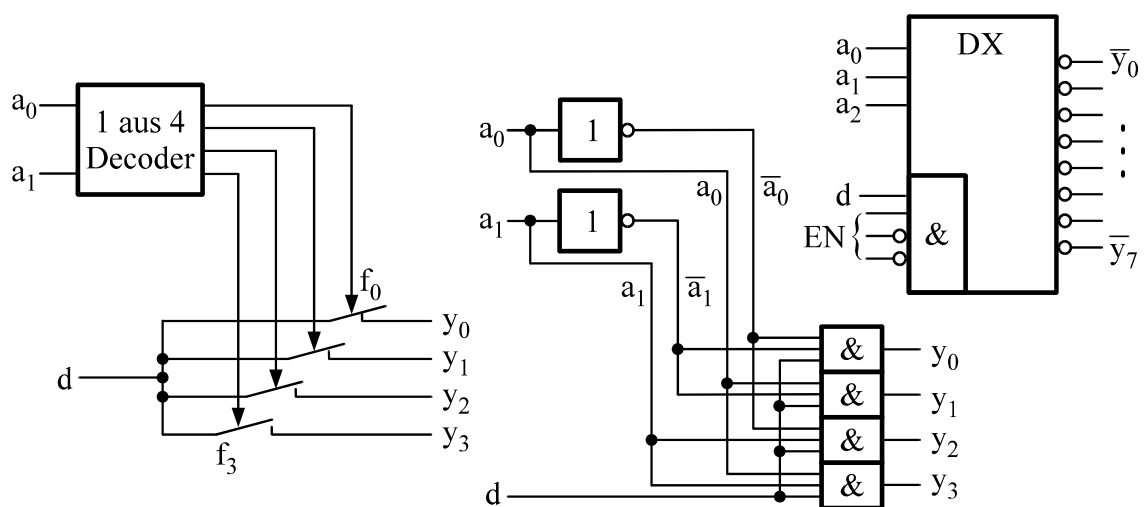


Bild 2.27: Demultiplexer

Die logische Funktion des Demultiplexers ist beschreibbar durch eine UND-Verknüpfung des Dateneingangssignals d mit den jeweiligen die Ausgangsadresse bildenden Mintermen des 1-aus-N-Decoders: z. B. $y_1 = a_0 \wedge \bar{a}_1 \wedge d$

Ein *Multiplexer* bildet die Umkehrung des Demultiplexers. Wahlweise können Eingangsdaten auf einen Ausgang gelegt werden. Wiederum wird mit Hilfe eines 1-aus-N-Decoders adressiert, hier der auf den Ausgang zu schaltende Eingang (vgl. Schaltnetz).

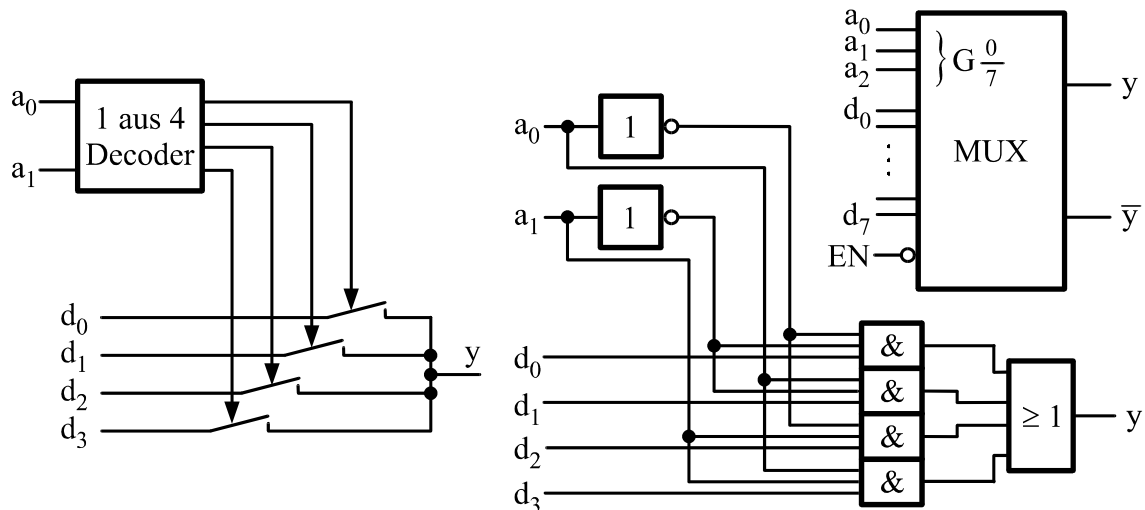


Bild 2.28: Multiplexer

Die logische Funktion des Multiplexers ist gegeben durch:

$$y = (\bar{a}_0 \wedge \bar{a}_1 \wedge d_0) \vee (a_0 \wedge \bar{a}_1 \wedge d_1) \vee (\bar{a}_0 \wedge a_1 \wedge d_2) \vee (a_0 \wedge a_1 \wedge d_3)$$

2.4.4 Schaltnetze mit Multiplexern

Im vorangehenden Abschnitt wurde die Funktion von Multiplexern und ihre Anwendung als Datenwegschaltung beschrieben. Die UND-Verknüpfung der Steuereingänge mit den Dateneingängen hat adressierende Eigenschaften. Jede Kombination der Steuervariablen wird mit einem Dateneingang UND-Verknüpft und auf den Ausgang durchgeschaltet. Dadurch wird ein Schaltnetz realisiert, das die Steuereingänge als Eingangsvariable hat. An den Dateneingängen liegen die Funktionswerte der Ausgangsvariablen.

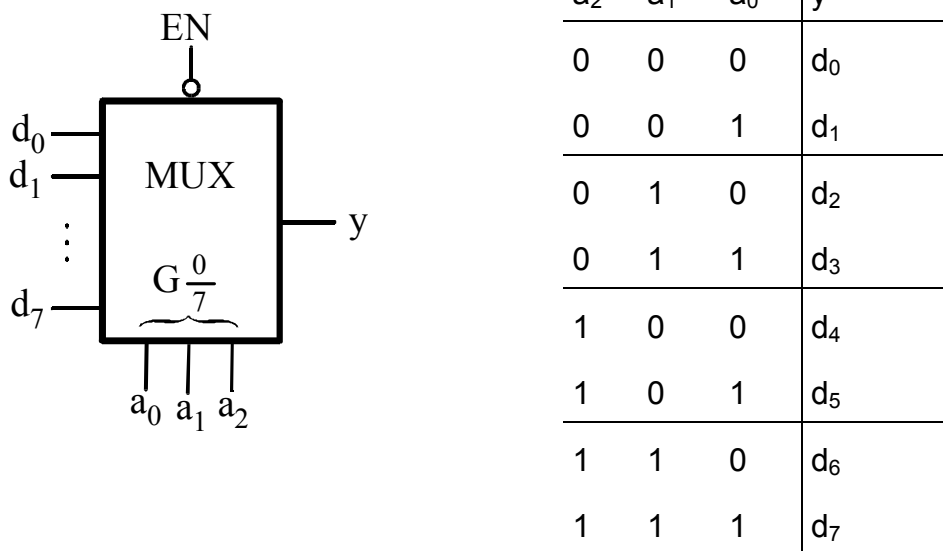
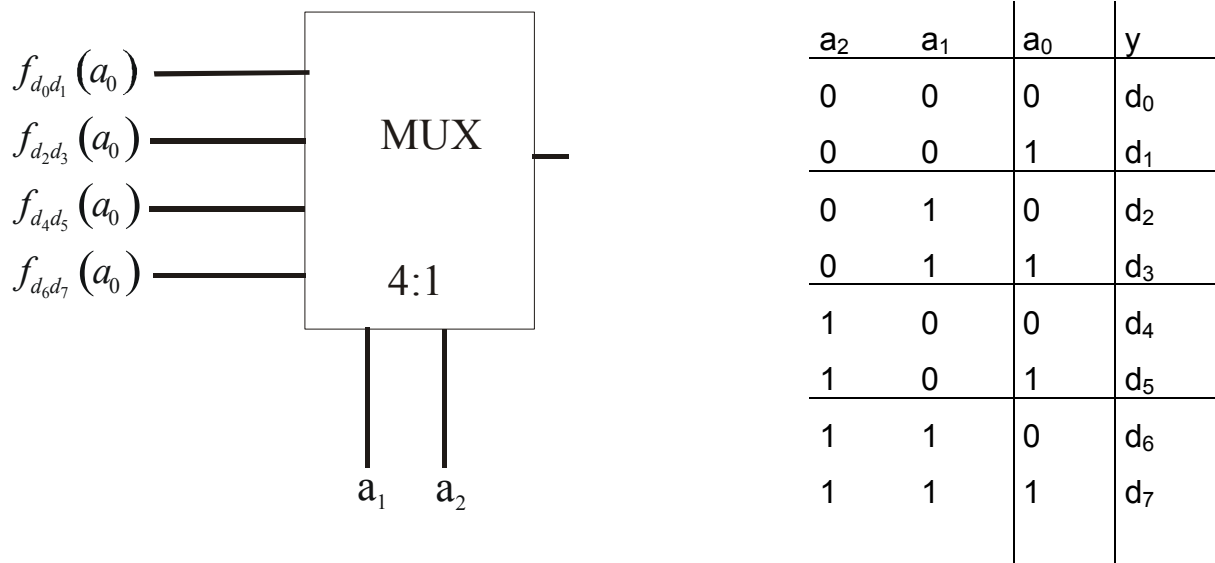
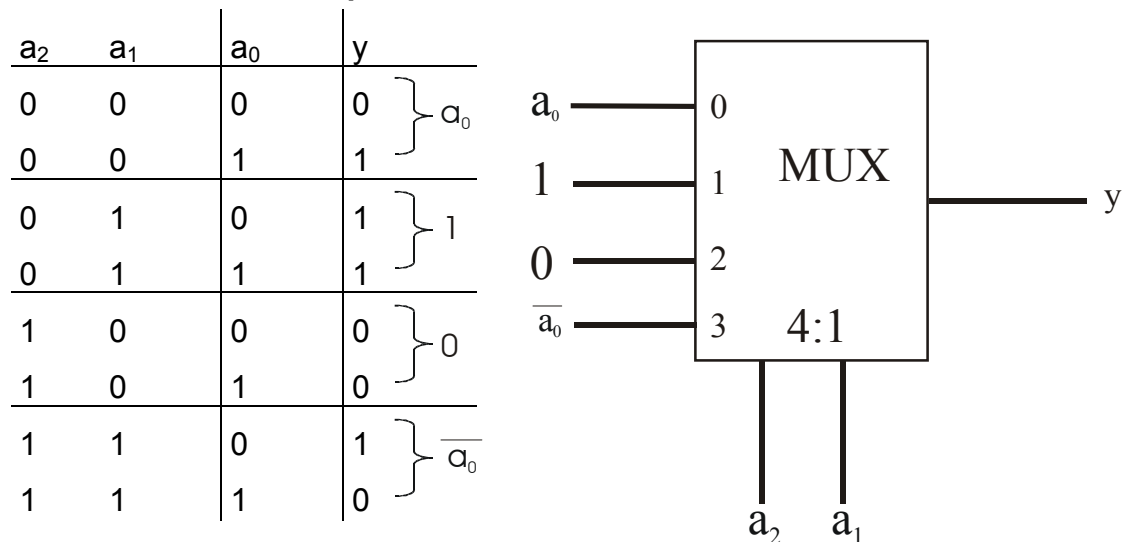


Bild 2.29: Beispiel: 8:1 Multiplexer

Ein N:1-Multiplexer lässt sich auch durch einen (N-1):1 Multiplexer ersetzen, also statt eines 8:1 wird ein 4:1-Multiplexer verwendet. Das bedeutet eine wesentliche Vereinfachung. Dieser Vereinfachung liegt folgende Überlegung zugrunde: die Adresse und der zugehörige Dateneingang sind Und-verknüpft. Es ist also möglich die Eingangsvariablen *aufzuspalten*, zwei(hier a_1, a_2) werden als Adresseingänge benutzt und die anderen werden den Dateneingängen zugeführt. Dabei wird die eigentliche Ausgangsvariable z.B y durch die dritte Steuervariable (a_0) oder durch die Konstanten 0 oder 1 ausgedrückt. Damit ergibt sich die Belegung der Dateneingänge für den Multiplexer. Schaltfunktionen lassen sich also mit Multiplexern vereinfachen. Die als Adresseingänge benutzten Eingangsvariablen werden aus der Schaltfunktion *eliminiert*. Jede Schaltfunktion mit n Eingangsvariablen kann durch einen Multiplexer mit 2^{n-1} Eingängen realisiert werden. An die Dateneingänge werden die Konstanten 0,1 oder die restliche Eingangsvariable angelegt. Soll die Schaltfunktion durch einen Multiplexer mit 2^{n-2} Eingänge realisiert werden, dann werden an die Dateneingänge die Konstanten 0,1, die restlichen Eingangsvariablen oder Verknüpfungen davon angelegt.

**Bild 2.30:** Vereinfachung durch 4:1 Multiplexer**Wahrheitstabelle, Beispiel****Bild 2.31:** Beispiel: 4:1 Multiplexer

Die Adresseingänge können aus den Eingangsvariablen frei gewählt werden. Jedoch ist es zweckmäßig, diejenigen Eingangsvariablen zu wählen, die in möglichst vielen Produkttermen der Schaltfunktion(DNF) vorkommen.

Die Realisierung von logischen Schaltungen mit Hilfe von Multiplexern ist technologisch von großer Bedeutung. Dies ist begründet dadurch, dass Multiplexer sich auf Basis von CMOS-Schaltungen (Transmissions- Gate, s. Abschnitt am Ende dieses Kapitels), aufwandgünstig realisieren lassen. Aufgrund dessen werden Schaltkreisfamilien für anwenderprogrammierbare Schaltungen (s. Kapitel 5) angeboten, die diese Eigenschaften nutzen.

2.5 CMOS- Grundsaltungen

Die jeweils verwendete Halbleitertechnologie hat unmittelbare Auswirkungen auf Eigenschaften und Einsatzgebiet der fertigen ICs.

Beurteilungskriterien für den Einsatz einer bestimmten Technologie sind u.a. die folgenden Punkte

- der erreichbare Integrationsgrad,
- die maximale Schaltgeschwindigkeit,
- die auftretende Verlustleistung,
- der Störabstand.

Prinzipiell sind für den Entwurf integrierter Schaltungen die in Bild 2.32 dargestellten Technologien einsetzbar.

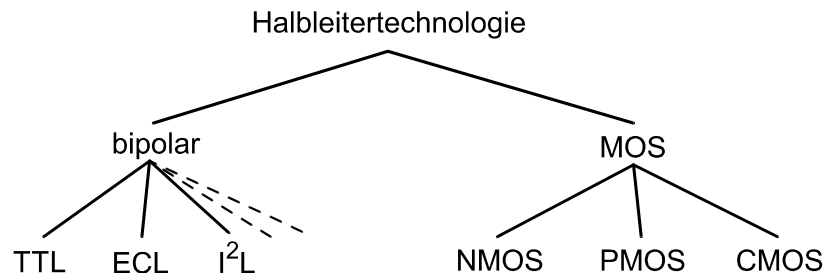


Bild 2.32: Überblick über verschiedenen Halbleitertechnologien

Die Halbleitertechnologien lassen sich in die zwei Hauptfamilien, die *Bipolar-Technologie* und die *MOS-Technologie* untergliedern. Der klassische Vorteil der Bipolar-Technologie, die hohe Schaltgeschwindigkeit, muß mit entsprechend hohen Verlustleistungen erkaufte werden. Ebenso ist der zu erreichende Integrationsgrad, d.h. die Anzahl der Transistoren bzw. Gatter pro Chip, begrenzt. Diese Begrenzung wird einerseits durch die im Vergleich zur MOS-Technologie aufwendigere Schaltungstechnik begründet, andererseits bereitet die Ableitung der Verlustwärme Schwierigkeiten. Dadurch ergeben sich technologische Grenzen der noch mit vertretbaren Kühlmaßnahmen handhabbaren Chipkomplexitäten.

Die MOS-Technologie bietet dagegen höhere Integrationsgrade und wesentlich geringere Verlustleistungswerte. Der entscheidende Durchbruch bei der MOS-Technologie gelang, als man die ursprünglich relativ niedrigen Schaltgeschwindigkeiten auf bipolar typische Werte anheben konnte.

Für den modernen Entwurf integrierter Schaltungen ist fast nur noch die CMOS-Technologie von Bedeutung. Einen Überblick zu den weiteren Technologie-Familien ist der einschlägigen Literatur zu entnehmen.

Die **CMOS**-Technologie (**C**omplementary **M**etal **O**xid **S**emiconductor) basiert darauf, daß stets zwei komplementäre Transistoren (ein n- und ein p-Kanal-Transistor) verwendet werden. Diese beiden komplementären Transistoren sind in Serie geschaltet und werden an ihren Gates vom gleichen Signal angesteuert. Hierdurch ist gewährleistet, dass (zumindest im statischen Betriebsfall) jeweils einer der beiden Transistoren gesperrt wird. Es fließt somit bis auf einen minimalen Sperrstrom kein Strom durch die Serienschaltung. Die CMOS-Technologie weist deshalb von allen Halbleitertechnologien die niedrigste Verlustleistung auf. Bild 2.33 stellt drei Grundelemente in CMOS-Technologie (Inverter, NAND, NOR) dar.

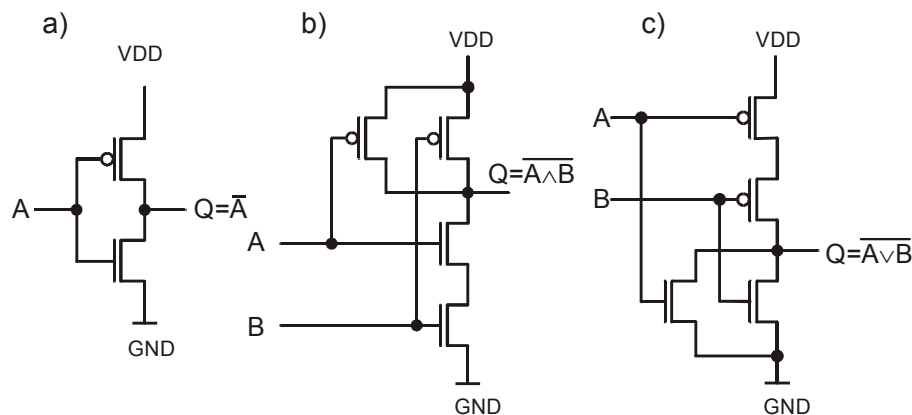


Bild 2.33: Inverter, NAND und NOR in CMOS-Technologie

Wie in Bild 2.33 dargestellt, werden bei der CMOS-Technologie die n-Kanal-Transistoren (sie bilden das sogenannte n-Netz) und die p-Kanal-Transistoren (das p-Netz) jeweils komplementär beschaltet. Sind z.B. wie in Bild 2.33 b) bei der Realisierung eines NAND-Gatters die beiden n-Kanal-Transistoren in Serie geschaltet, werden die beiden p-Kanal-Transistoren parallel geschaltet.

Den in Bild 2.33 dargestellten Implementierungen eines NAND- und eines NOR-Gatters kommt eine besondere Bedeutung zu, da NAND und NOR jeweils eine sogenannte *vollständige Verknüpfungsmenge* bilden. Dies bedeutet, daß jede beliebige Logikfunktion sich aus NAND- oder NOR-Gattern aufbauen läßt.

Allgemein kann ein vollkomplementäres Gatter mit mehreren Eingängen $x_1, x_2, \dots, x_n$ stets als einen Umschalter aufgefaßt werden, der unter bestimmten Bedingungen die Versorgungsspannung über einen Block aus PMOS-Transistoren (p-Netz) gegen den Ausgang schaltet bzw. die Masse über einen Block aus NMOS-Transistoren (n-Netz). Es muß dabei gelten, daß die beiden Blöcke zu keiner Zeit gleichzeitig leiten dürfen, da sonst die Versorgungsspannung kurzgeschlossen und dadurch die Schaltung unter Umständen zerstört würde. In Abhängigkeit von den in Bild 2.34 eingetragenen Eingangssignalen, die an beide Netze geführt sind, muß diese Bedingung für alle Kombinationen der Eingangssignale erfüllt sein.

Dabei ist zu beachten, daß die Polarität der Eingangssignale richtig angesetzt wird. Ein einstufiges Gatter in CMOS-Technik hat immer ein invertiertes Ausgangssignal. Daher muß das n-Netz so ausgelegt werden, daß wenn die zu realisierende boolesche Gleichung erfüllt ist, der Ausgang der Schaltung mit Masse über das n-Netz verbunden ist (positive Logik bei invertiertem Ausgangssignal). Gleichzeitig wird das p-Netz in negativer Logik ausgeführt, d.h. durch Invertierung der Gleichungen für das n-Netz und durch Anwendung der *de Morgan'schen Gesetze* erhält man ein p-Netz, das genau dann leitet (VDD mit dem Ausgang verbindet), wenn das n-Netz nicht leitet.

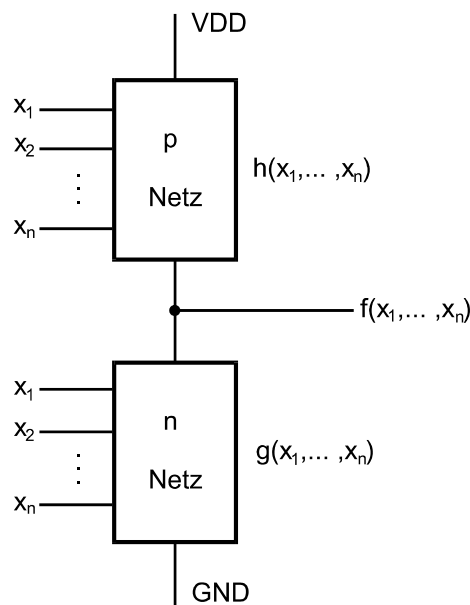


Bild 2.34: Prinzipbild einer Gatterlogik

Dieser Zusammenhang soll an einem einfachen Beispiel verdeutlicht werden. Realisiert werden soll eine Logikfunktion

$$f_{\text{Res}}(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \vee (x_3 \wedge x_4) \quad .$$

Da f hier nicht invertiert vorliegt, kann die Schaltung z.B. aus einem invertierten einstufigen Gatter mit der Funktion $\bar{f}$ sowie einem weiteren Inverter zusammengesetzt werden.

Das einstufige Gatter für das n-Netz in positiver Logik *bei invertiertem Ausgangssignal* erfüllt dabei die Funktion

$$\bar{f}_n(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \vee (x_3 \wedge x_4) \quad .$$

Die Transistoranordnung für das n-Netz (bei invertiertem Ausgangssignal) ergibt sich jetzt wie zuvor schon in Bild 2.33 dargestellt, indem die AND (NAND)-Verknüpfung

durch eine Reihenschaltung und die OR (NOR)-Verknüpfung durch eine Parallelschaltung der n-Kanal-Transistoren realisiert wird.

Die Transistoranordnung des p-Netzes in negativer Logik *bei nicht invertiertem Ausgangssignal* ergibt sich, indem die Gleichung invertiert wird, die de Morgan'schen Gesetze angewendet werden und die boolesche Gleichung solange umgeformt wird, bis jedes Eingangssignal invertiert in die Funktion eingeht. Es folgt dann daraus

$$\begin{aligned} f_p(x_1, x_2, x_3, x_4) &= \overline{(x_1 \vee x_2) \vee (x_3 \wedge x_4)} \\ &= \overline{(x_1 \vee x_2)} \wedge \overline{(x_3 \wedge x_4)} \\ &= \overline{x_1} \wedge \overline{x_2} \wedge (\overline{x_3} \vee \overline{x_4}) . \end{aligned}$$

Diese Gleichung entspricht der in negativer Logik zu realisierenden Transistoranordnung für die p-Kanal-Transistoren. Berücksichtigt man, daß ein PMOS-Transistor genau dann leitet, wenn sein Gate ein negatives Potential gegenüber der Versorgungsspannung VDD hat, so ergibt sich für die im p-Netz zu realisierende Funktion wieder eine Verschaltung, bei der eine UND-Verknüpfung eine Reihenschaltung und eine OR-Verknüpfung eine Parallelschaltung ist, dies jetzt nur in negativer Logik. Insgesamt ergibt sich für die Gesamtschaltung, daß das p-Netz genau dann leitet, wenn das n-Netz nicht leitet. In Bild 2.35 ist die resultierende Schaltung für die Funktion f bestehend aus n- und p-Netz dargestellt.

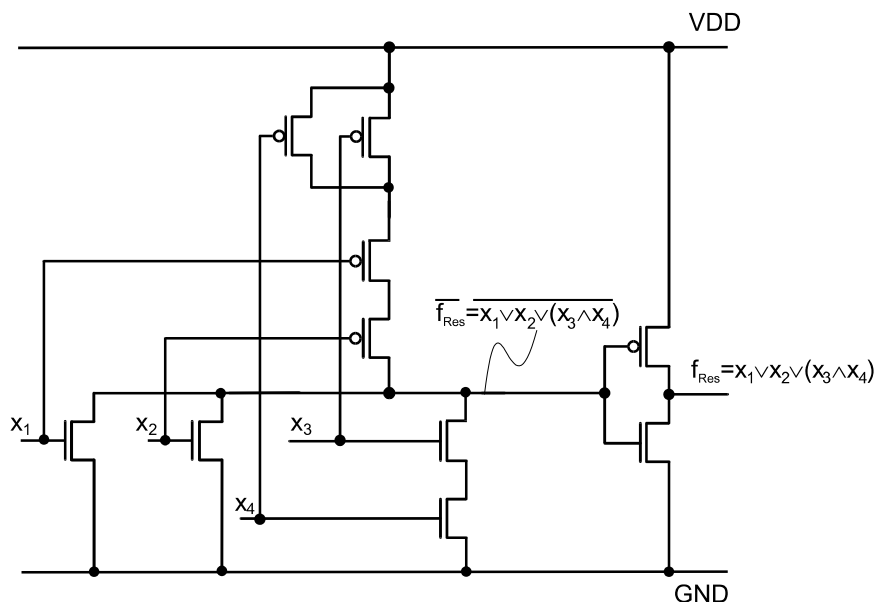


Bild 2.35: Resultierende Schaltung für die Funktion f_{Res} in CMOS-Technologie

Für die Herstellung von CMOS-Schaltungen existiert eine Vielzahl von Technologien (n-Wannen-Prozeß, p-Wannen-Prozeß usw., siehe hierzu z.B. [Hilleri96]).

Neben der bisher vorgestellten statischen Gatterlogik gibt es weitere Logikformen (Strukturen), die sich durch Eigenschaften wie eine verringerte Transistoranzahl

auszeichnen. Zu diesen Sonderformen zählen z.B. die *Pseudo-NMOS-Logik* (ein einziger p-Kanal-Transistor als Pull-up-Pfad) oder die *dynamische CMOS-Logik* (für getaktete Systeme unter Verwendung eines reinen n-Netzwerkes), siehe hierzu z.B. [Pirsch96] oder [Reifsch98].

Einige wichtige Grundelemente der vorgestellten statischen Gatterlogik, die auch im weiteren Verlauf dieses Buches verwendet werden, sollen hier exemplarisch aufgeführt werden.

2.5.1.1 Transmission-Gate

Die Funktion eines Transmission-Gates¹ besteht darin, daß Eingang und Ausgang dieses Elementes entweder niederohmig verbunden oder getrennt werden. Beide Anschlüsse sind dabei gleichberechtigt. Dies bedeutet, daß ein Transmission-Gate als bidirektionales Übertragungsglied verwendet werden kann. Das eigentliche Transmission-Gate besteht dabei aus einer Parallelschaltung eines n- und eines p-Kanal-Transistors. Bild 2.36 stellt den Aufbau eines Transmission-Gates sowie sein logisches Ersatzschaltbild als steuerbarer Verbindungsschalter dar.

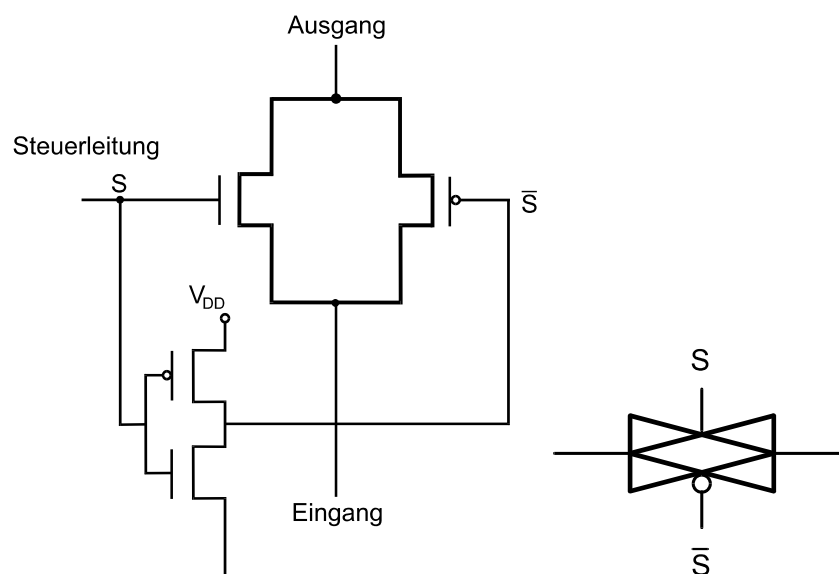


Bild 2.36: Aufbau und Ersatzschaltbild eines Transmission-Gates

Im Unterschied zu den bisher diskutierten logischen Gattern tritt keine Pegelregenerierung bei einem Transmission-Gate auf. Der Störabstand des übertragenen Signals wird daher bei einer Kette von Transmission-Gates umso schlechter, je mehr Elemente zusammenschaltet werden. Man verwendet daher Transmission-Gates nur in Verbindung mit konventionellen Gattern.

¹ In der deutschsprachigen Literatur z.T. als "Transmissions-Glied" bezeichnet.

2.5.1.2 Multiplexer

Die zuvor beschriebenen Transmission-Gates können sehr gut zur Implementierung von Multiplexern verwendet werden. Ein Multiplexer hat die Aufgabe, ein durch Steuerleitungen adressierbares Eingangsdatum aus einer Reihe von Eingangsleitungen auszuwählen und auf den Ausgang durchzuschalten. Die logische Funktion eines 4:1 Multiplexers ist beispielsweise durch

$$y = \bar{s}_0 \bar{s}_1 d_0 \vee s_0 \bar{s}_1 d_1 \vee \bar{s}_0 s_1 d_2 \vee s_0 s_1 d_3$$

gegeben. Bild 2.37 zeigt hierzu die prinzipielle Wirkungsweise sowie die Implementierung dieses 4:1 Multiplexers basierend auf Transmission-Gates.

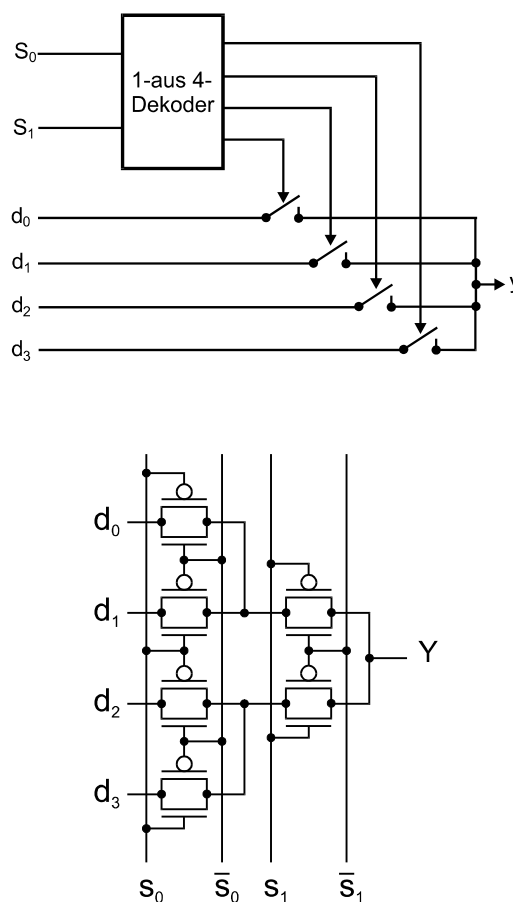


Bild 2.37: Prinzip und Implementierung eines 4:1 Multiplexers mit Transmission-Gates

Eine Implementierung der logischen Funktion dieses Multiplexers ist darüber hinaus natürlich auch mit logischen Gattern möglich. Der Vorteil bei der Verwendung von Transmission-Gates liegt darin, daß die Signalübertragung bidirektional ist, was dazu führt, daß der Multiplexer identisch mit dem Demultiplexer ist, bei dem durch die Steuervariablen ausgewählt werden kann, an welchen Ausgang die Eingangsinformation weitergeleitet werden soll.

2.5.2 Literaturhinweise

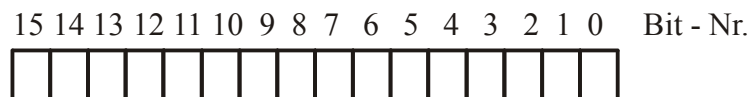
- [MeadCon80] Mead, C. Conway, L. "Introduction to VLSI Systems", Addison-Wesley, 1980
- [Schröder, H.; Blume, H.] Mehrdimensionale Signalverarbeitung 2, Teubner-Verlag, 2000
- [Salcic, Z.; Smailagic, A.] Digital Systems Design and Prototyping using Field Programmable Logic
- [MAX7000] Altera Programmable Logic Device Family,
<http://www.altera.com/literature/ds/m7000.pdf>
- [Goser91] Goser, K. : "Großintegrationstechnik", Band I, Hüthig Verlag, 1991
- [Hilleri96] *Hilleringmann, U.* : "Silizium Halbleitertechnologie", Teubner-Verlag, Stuttgart, 1996
- [Klar93] *Klar, H.* : "Integrierte digitale Schaltungen MOS/BICMOS", Springer Verlag, 1993
- [Pirsch96] *Pirsch, P.* : "Architekturen der digitalen Signalverarbeitung", Verlag B.G. Teubner Stuttgart, ISBN 3-519-06157-0, 1996

Kapitel 3

Arithmetisch-logische Einheiten

3.1 Digitale Zeichen- und Zahlendarstellungen

Eine zweiwertige Information (0 bzw. 1, wahr bzw. falsch) wird Bit genannt, eine Gruppe von 8 Bit bezeichnet man als Byte. Verwendet werden im allgemeinen Wortlängen zur Zahlendarstellung zwischen 16 bzw. 32 (64) Bit. Für die Zählung der Bitstellen wird hier folgendes Schema verabredet (von rechts nach links mit Null beginnend):



Das Bit mit der geringsten Wertigkeit wird dabei auch als LSB, das mit der höchsten Wertigkeit als MSB bezeichnet (**L**east/**M**ost **S**ignificant **B**it). Ganze Zahlen im 16 Bit-Format werden mit einem Bit für das Vorzeichen (0:+, 1:-) dargestellt:

Der verfügbare Bereich ist dann

$$\begin{aligned} -2^{15} &\leq N \leq +2^{15} - 1 \\ -32768 &\leq N \leq 32767 \end{aligned}$$

Einen entsprechenden größeren Zahlenbereich deckt man ab durch Zusammenfügen zweier Einzelwörter zu einem Doppelwort (Doppelwortgenauigkeit). Der verfügbare Zahlenbereich ist dann

$$\begin{aligned} -2^{31} &\leq N \leq +2^{31} - 1 \\ -2.147 \cdot 10^9 &\leq N \leq +2.147 \cdot 10^9 \end{aligned}$$

Doppelwortformate werden insbesondere zur Darstellung von Gleitpunktzahlen herangezogen, z.B. für "Einfache" Genauigkeit nach dem IEEE-Standard (IEEE-P754-Floating-Point-Standard), entsprechend der Definition (Bild 3.1).

$$Z_{GL} = (-1)^v \cdot m \cdot 2^e$$

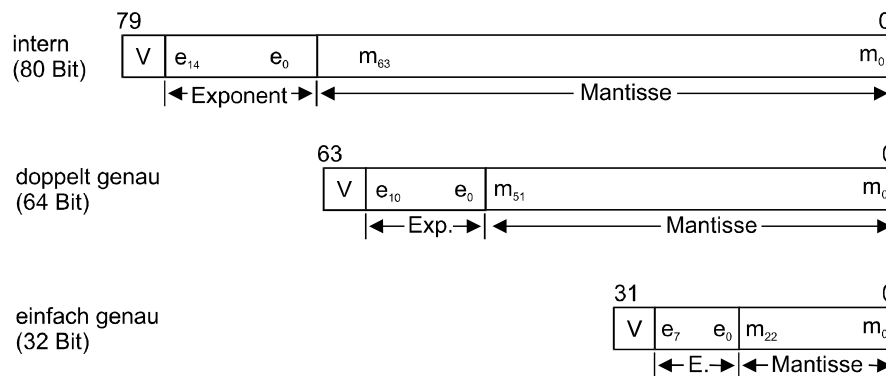


Bild 3.1: Gleitkomma-Darstellung im IEEE-Floating-Point-Standard (P 754)

Der Exponent e wird dabei im "Offset-Binary"-Code dargestellt und bewegt sich bei 8 Bit Genauigkeit im Bereich

$$0.0000000 \leq e \leq 1.1111111$$

$$-128 \leq e \leq 127$$

Die Mantisse (Betrag-Vorzeichen-Darstellung) wird auf 1 normiert, so daß das nicht mitgeführte Mantissenbit m_{23} stets 1 ist. Der Betrag der Mantisse bewegt sich dann zwischen $1 \leq m < 2$. Der erreichbare Zahlenbereich der Beträge $|N|$ ist durch den Exponenten gegeben und liegt zwischen

$$2^{-128} \leq |N| \leq 2^{127}$$

Die Mantissengenauigkeit ist 24 Bit (incl. Vorzeichen). Die Zahlengenauigkeit ist damit über den Zahlenbereich nicht konstant, sondern exponentenabhängig.

3.2 Digitale Zahlendarstellungen, Festkomma- und Ganzzahlenformate

Für die Realisierung von Rechenschaltungen sind verschiedene Zahlendarstellungen gebräuchlich. Ihre Auswahl ist abhängig von der Effizienz der auszuführenden Operationen. Gesichtspunkte sind z. B. dabei

- Geschwindigkeit hinsichtlich des Datendurchsatzes
- Genauigkeit der auszuführenden Operation
- Aufwand in der Realisierungsstruktur.

Für schnelle Signalverarbeitungstechniken und für aufwandsarme Realisierungen sind nach wie vor Festkommadarstellungen bzw. Ganzzahlendarstellungen unentbehrlich. Es werden jedoch zukünftig in Anwendungen, die eine hohe

Signaldynamik aufweisen, auch Gleitkommadarstellungen zunehmend eingesetzt. Hier werden Festkommadarstellungen betrachtet, da sie die Grundlage bilden für arithmetische Rechenbausteine. Die gilt auch für Gleitkommaoperatoren, die ja für Mantisse und Exponenten getrennt Festkommazahlenmodule enthalten müssen.

Zahlensysteme mit einer ganzzahligen Basis B (z. B. 10 im Dezimalsystem oder 2 im Dualsystem) und einem begrenzten Ziffernvorrat $b_i < B$ (z. B. $\{0, 1, \dots, 9\}$ im Dezimalsystem oder $\{0,1\}$ im Dualsystem) bezeichnet man als "polyadische Zahlensysteme".

Eine Eigenschaft solcher Zahlensysteme ist die Stellenbewertung. Ist eine Zahl etwa in der Form $b_{N-1} \dots b_2 b_1 b_0$ gegeben, so hängt der Wert jeder Ziffer von der Position in der Ziffernreihe ab: Man ordnet jeder Position einen von rechts nach links steigenden Stellenwert zu.

Für eine natürliche Zahl im Dualzahlensystem gilt bei N -stelliger Darstellung für ihren Wert im Dezimalsystem Z_n

$$Z_n = \sum_{i=0}^{N-1} b_i \cdot 2^i = b_{N-1} \cdot 2^{N-1} + \dots + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

Eine gebrochene Zahl wird im Dualzahlensystem bei M -stelliger Darstellung zu

$$Z_b = \sum_{i=-M}^{-1} b_i \cdot 2^i = b_{-1} \cdot 2^{-1} + b_{-2} \cdot 2^{-2} \dots + b_{-M} \cdot 2^{-M}.$$

Bei einer Festkomma-Dualzahl werden ähnlich wie beim Dezimalbruch die Stellenwerte rechts vom Komma als negative Zweierpotenzen aufgefasst. Die Stellenzahl ist hier $N+M$:

$$\begin{aligned} Z_{FK} &= \sum_{i=-M}^{N-1} b_i \cdot 2^i \\ &= b_{N-1} \cdot 2^{N-1} + \dots + b_1 \cdot 2^1 + b_0 \cdot 2^0 + b_{-1} \cdot 2^{-1} + \dots + b_{-M} \cdot 2^{-M} \end{aligned}$$

Eine wichtige Rolle bei der binären Zahlendarstellung mit Blick auf die verschiedenen auszuführenden Operationen spielt die Darstellung negativer Zahlen. Verschiedene Codiertechniken sind hierfür gebräuchlich und sind zunächst in einer Übersicht dargestellt:

- Betrag - Vorzeichen - Darstellung Z_{BV}
- Offset - Dual - (Binary Offset -) Darstellung Z_{BO}
- Zweier - Komplement - Darstellung Z_{2K}
- Einer - Komplement - Darstellung Z_{1K}

Meistens werden hierbei Ganzzahldarstellungen oder gebrochene Zahlen verwendet. Der Übergang zwischen diesen Darstellungen erfolgt durch Stellenverschiebung bzw. Multiplikation mit der entsprechenden Zweierpotenz.

Dezimal- zahlen- wert	Betrag- Vorzeichen Z_{BV}	Binary- Offset Z_{BO}	1er Komplement Z_{1K}	2er Komplement Z_{2K}
+7	0 1 1 1	1 1 1 1	0 1 1 1	0 1 1 1
+6	0 1 1 0	1 1 1 0	0 1 1 0	0 1 1 0
+5	0 1 0 1	1 1 0 1	0 1 0 1	0 1 0 1
+4	0 1 0 0	1 1 0 0	0 1 0 0	0 1 0 0
+3	0 0 1 1	1 0 1 1	0 0 1 1	0 0 1 1
+2	0 0 1 0	1 0 1 0	0 0 1 0	0 0 1 0
+1	0 0 0 1	1 0 0 1	0 0 0 1	0 0 0 1
+0	0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 0
-0	1 0 0 0	-	1 1 1 1	-
-1	1 0 0 1	0 1 1 1	1 1 1 0	1 1 1 1
-2	1 0 1 0	0 1 1 0	1 1 0 1	1 1 1 0
-3	1 0 1 1	0 1 0 1	1 1 0 0	1 1 0 1
-4	1 1 0 0	0 1 0 0	1 0 1 1	1 1 0 0
-5	1 1 0 1	0 0 1 1	1 0 1 0	1 0 1 1
-6	1 1 1 0	0 0 1 0	1 0 0 1	1 0 1 0
-7	1 1 1 1	0 0 0 1	1 0 0 0	1 0 0 1
(-8)		(0 0 0 0)		(1 0 0 0)

Tabelle 2.1: Binäre Zahlendarstellungen mit Vorzeichen

3.2.1 Das Betrag-Vorzeichen-Format

Eine naheliegende und einfache Darstellungsform für positive und negative Zahlen ist die *Betrag-Vorzeichen-Darstellung*. Die Codierung des Vorzeichens b_N erfolgt in der Regel

$$b_N = 0 \text{ für } Z_{BV} \geq 0$$

$$b_N = 1 \text{ für } Z_{BV} < 0$$

Bei einer z. B. $N+1$ - stelligen Darstellung dient ein Bit, das MSB, zur Vorzeichendarstellung. Die anderen Bits kennzeichnen den Betrag.

Beispiel: Achtstellige Betrag-Vorzeichen-Zahl, $Z_{BV} = -101$

b_N	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	0	0	1	0	1

Hieraus ergibt sich als *Wert einer Betrag-Vorzeichen-Zahl*:

$$Z_{BV} = (-1)^{b_N} \cdot \sum_{i=0}^{N-1} b_i \cdot 2^i$$

Für die Ausführung von Multiplikationen und Divisionen, bei denen das Vorzeichen getrennt behandelt werden kann, ist diese Darstellung gut geeignet. Ungünstig ist die Subtraktion auszuführen. Hier ist wegen des möglichen Vorzeichenwechsels eine besondere Steuerlogik erforderlich.

3.2.2 Das Zweierkomplement-Format

Bei der *Zweierkomplement-Darstellung* kann dieser zusätzliche Aufwand für die Subtraktion vermieden werden. Der Gedanke der Komplementbildung ergibt sich daraus, daß negative Zahlen so codiert werden sollen, daß eine Subtraktion beliebiger Zahlen stets auf eine Addition negativer Zahlen zurückgeführt werden kann. Man kommt zum Zweierkomplement durch folgende Überlegung:

Wiederum werden positive Zahlen mit einem Vorzeichenbit 0 und negative Zahlen mit Vorzeichenbit 1 verabredet. Gegeben sei eine positive Dualzahl Z_{N1} mit einem Betrag B_{N1} in z. B. (N=4)-stelliger Darstellung. Z_{N1} hat dann mit Vorzeichenbit (hier = 0) N+1=5 Stellen:

	2^4	2^3	2^2	2^1	2^0
$Z_{N1}=5=$	0	0	1	0	1
$B_{N1}=5=$		0	1	0	1

Im Zweierkomplement wird nun die negative Zahl $Z_{N2} = -B_{N1}$ dargestellt durch die Dualzahl

$$B_{N2} = 2^N - B_{N1}$$

Daraus ergibt sich mit dem Vorzeichen (hier = 1) die Zahl Z_{N2} , im Beispiel gilt:

2^N	1	0	0	0	0
$-B_{N1} -$		0	1	0	1
B_{N2}		1	0	1	1
Z_{N2}	1	1	0	1	1

$\triangleq B_{N2}$ und Vorzeichenbit

Eine Addition der beiden Zahlen B_{N1} und B_{N2} , die einer Subtraktion entspricht, führt demnach auf den Zahlenwert 2^N . Bezieht man die Vorzeichenbits mit ein, addiert also Z_{N1} und Z_{N2} , so entsteht im definierten Zahlenbereich - (N+1) Bit - der gewünschte Wert 0, Überträge in höherwertigen Stellen werden nicht berücksichtigt. Im Beispiel gilt:

	2^N			2^1	2^0
B_{N1}		0	1	0	1
$+B_{N2} +$		1	0	1	1
2^N	1	0	0	0	0
Z_{N1}	0	0	1	0	1
$Z_{N2} +$	1	1	0	1	1
0	0	0	0	0	0

Allgemein lässt sich eine (N + 1 - stellige Zweierkomplementzahl) durch folgenden Ausdruck darstellen:

$$Z_{2K} = -b_N \cdot 2^N + \sum_{i=0}^{N-1} b_i \cdot 2^i$$

$$= -b_N \cdot 2^N + b_{N-1} \cdot 2^{N-1} + \dots + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

Zwei Zahlenbeispiele: Achtstellige 2er Komplementzahl

$$Z_{2K} = +101$$

$$Z_{2K} = -101$$

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
+101	0	1	1	0	0	1	0	1
-101	1	0	0	1	1	0	1	1

Der verfügbare Zahlenbereich ist durch die $N+1$ Stellen gegeben und erstreckt sich über

$$-2^N \leq Z_{2K} \leq 2^N - 1$$

Bei einer festen Wortlänge von z. B. $N+1$ Bit erweist sich das 2er Komplement als ein mit 2^{N+1} Schritten, d. h. mit der Modulofunktion $\text{mod } 2^{N+1}$, zyklischer Code. Man kann deshalb anschaulich die 2er Komplementdarstellung im Zahlenring darstellen:

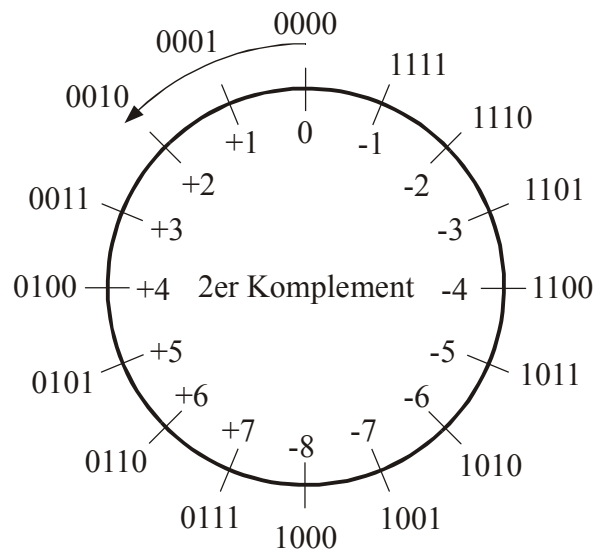


Bild 3.2: Zahlenkreisdarstellung für das 2er Komplement, 4 Bit Wortbreite

Die *Komplementierung einer gegebenen Zahl*, d. h. der *Vorzeichenwechsel*, erfolgt wie ausgeführt nach der Beziehung

$$B_{N2} = 2^N - B_{N1}$$

durch die die gekennzeichnete 0,1-Kombination (ohne Vorzeichen) ermittelt wird.

Hierfür gilt auch:

$$\sum_{i=0}^{N-1} \underbrace{b_{i2}}_{\text{gesuchte Bit-kombination}} \cdot 2^i = 2^N - \sum_{i=0}^{N-1} \underbrace{b_{i1}}_{\text{gegebene Bit-kombination}} \cdot 2^i$$

Da gilt:

$$\sum_{i=0}^{N-1} b_{i1} \cdot 2^i + \sum_{i=0}^{N-1} \bar{b}_{i1} \cdot 2^i = \sum_{i=0}^{N-1} 2^i (\hat{=} 11111 \dots) = 2^N - 1$$

$$\sum_{i=0}^{N-1} b_{i1} 2^i = 2^N - 1 - \sum_{i=0}^{N-1} \bar{b}_{i1} \cdot 2^i$$

und es folgt für die gesamte Komplementbildung

$$\begin{aligned}
 B_{N2} &= \sum_{i=0}^{N-1} b_{i2} \cdot 2^i = 2^N - \sum_{i=0}^{N-1} b_{i1} \cdot 2^i \\
 &= 2^N - 2^N + 1 + \sum_{i=0}^{N-1} \bar{b}_{i1} \cdot 2^i \\
 &= \sum_{i=0}^{N-1} \bar{b}_{i1} \cdot 2^i + 1
 \end{aligned}$$

Als Regel folgt, dass zur Zweierkomplementbildung (Vorzeichenwechsel) die betrachtete Zahl in ihren Bits logisch negiert werden müssen und zusätzlich eine 1 ($= 2^0$) zu addieren ist.

Schließlich ist noch das neue Vorzeichen zu ergänzen, vgl. Beispiel:

B_{N1}	0	1	0	1
$\bar{B}_{N1}$	1	0	1	0
+1				1
B_{N2}	1	0	1	1
Z_{N2}	1	1	0	1

Ersichtlich kann auch hier das Vorzeichenbit mit einbezogen werden, vgl. Beispiel:

Z_{N1}	0.	0	1	0	1
$\bar{Z}_{N1}$	1	1	0	1	0
+1					1
Z_{N2}	1	1	0	1	1

Mit der Einführung der 2er - Komplementdarstellung kann innerhalb des durch die Wortlänge definierten Zahlenbereiches die *Addition positiver und negativer Zahlen* mit einem Additionswerk vorgenommen werden. Eine Subtraktion

$$\begin{aligned}
 D_{K2} &= Z_{2K1} - Z_{2K2} \\
 &= Z_{2K1} + (-Z_{2K2})
 \end{aligned}$$

kann durch die oben beschriebene Komplementierung des zu subtrahierenden Wertes auf eine Addition zurückgeführt werden. Eine Schaltung für ein 4-Bit-Segment zeigt das folgende Bild.

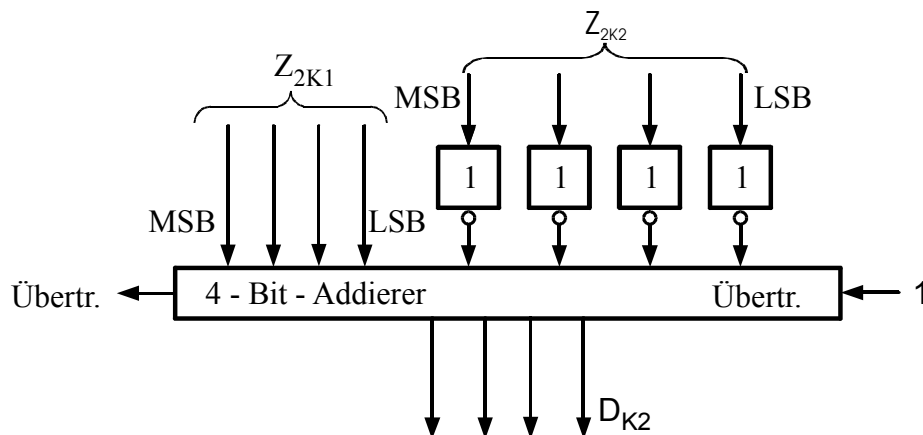


Bild 3.3: Schaltung zur Subtraktion zweier 4-Bit-Zahlen

Die Zahl Z_{2K2} wird komplementiert, d. h. alle Bits werden komplementiert und über den Übertragseingang wird eine 1 in der Wertigkeit des LSB addiert (LSB, MSB - Least, Most Significant Bit).

3.3 Addierer-Schaltnetze

Ziel dieses Abschnitts ist es, die für eine Schaltung zur digitalen Signalverarbeitung notwendigen elementaren Komponenten zu erläutern. Zum Aufbau derartiger Architekturen gehören insbesondere:

- arithmetische Bausteine zur Addition (bzw. Subtraktion bei negativen Koeffizienten)
- arithmetische Komponenten zur Multiplikation
- Koeffizientenspeicher, Register
- Schaltungen zur Rundung und Überlaufkontrolle
- Ablaufsteuerungen

Von besonderem Interesse sind dabei die arithmetisch-logischen Bausteine zur Addition und Multiplikation, da lineare und nichtlineare Signalverarbeitungsschaltungen besonders häufig von diesen Elementen Gebrauch machen. In diesem Zusammenhang sind *Schaltketten* (siehe Bild 3.4) als Boolesche Grundbausteine von grundsätzlicher Bedeutung. Eine Schaltkette ist die technische Realisierung eines Systems von Booleschen Funktionsbündeln

$$\bar{u}_{i+1} = f(\bar{u}_i, \bar{e}_i) \quad (3.1)$$

und gleichzeitig ggf.

$$\vec{a}_i = g(\vec{u}_i, \vec{e}_i) \quad (3.2)$$

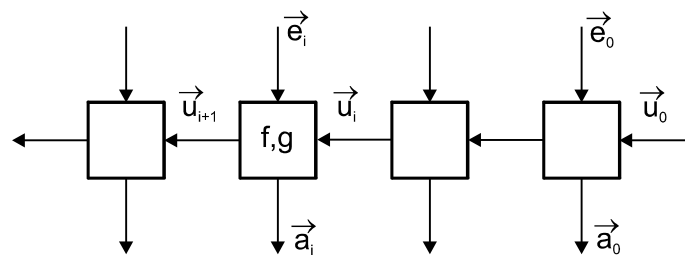


Bild 3.4: Schaltkette

Die Schaltkette besteht somit aus einer Kaskade von gleichartigen Schaltnetzen (siehe Bild 3.4). Mit Hilfe von Schaltketten lassen sich digitale Datenwörter logisch oder arithmetisch verarbeiten, z.B. Zahlenvergleiche oder Additionen durchführen. Grundsätzlich bilden Schaltketten die Basis für alle Operationen, die parallel auf digitale Wörter, d.h. bitparallel, angewendet werden. Sie bilden daher Grundbausteine für Rechenwerke, die eine Verkettung von Zellen zur Kaskadierung von Operationen und zur Übertragbehandlung erfordern.

3.3.1 Addierer-Schaltnetze mit Übertragsweiterleitung

Die Addition zweier Dualzahlen erfolgt nach den Rechenregeln der Arithmetik formal wie im Dezimalsystem. Es werden Bits gleicher Wertigkeiten unter Berücksichtigung von Überträgen aus niederwertigeren Operationen addiert. Dabei entstehen Überträge für höherwertige Operationen, die bei den entsprechenden Addierschritten zu berücksichtigen sind. Gegeben seien zwei Dualzahlen A und B, die entsprechend dem angegebenen Schema zu addieren sind :

A:	a_N	...	a_i	...	a_1	a_0	Dualzahl A
B:	b_N	...	b_i	...	b_1	b_0	Dualzahl B
C:	c_N	c_{i+1}	c_i	...	c_1	c_0	Überträge
D:	s_N	...	s_i	...	s_1	s_0	Summe

(3.3)

Die Aufgabe besteht darin, das angegebene Schema in einen Algorithmus von Teiloperationen umzusetzen. Dabei ist die Möglichkeit gegeben, eine räumlich parallele Realisierung von Teiloperationen durchzuführen, also eine Umsetzung des Algorithmus in eine Architektur parallel arbeitender Zellen vorzunehmen. Andererseits kann in Verbindung mit einem Rechenwerk und einem Steuerwerk

auch eine zeitlich sequentielle Abarbeitung des Algorithmus über ein geeignetes Mikroprogramm vorgenommen werden.

Schaltnetze zur Realisierung einer Mikrooperation für die Durchführung einer binären Addition sind der Volladdierer und der Halbaddierer, die aus den Eingangssignalen (Booleschen Variablen) a_i und b_i das Ausgangssignal s_i (Summenbit) bzw. in Verbindung mit einem Eingangsübertrag c_i zusätzlich ein Übertragsbit c_{i+1} erzeugen.

Den Zusammenhang zwischen arithmetischer Funktion des Volladdierers und seiner Booleschen Repräsentation kann man aus seiner Wertetabelle (Tabelle 3.2) entnehmen.

Eingangsbits			interne Bits			Ausgangsbits	
c_i	b_i	a_i	p_i	g_i	r_i	s_i	c_{i+1}
0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	0
0	1	0	1	0	0	1	0
0	1	1	0	1	0	0	1
1	0	0	0	0	0	1	0
1	0	1	1	0	1	0	1
1	1	0	1	0	1	0	1
1	1	1	0	1	0	1	1

Tabelle 3.2: Wahrheitstabelle des Volladdierers

Der Volladdierer erzeugt das Summenbit s_i mit LSB als "least significant bit"

$$\begin{aligned}
 s_i &= LSB[a_i + b_i + c_i] = LSB[p_i + c_i] \\
 &= a_i \bar{b}_i \bar{c}_i \vee \bar{a}_i b_i \bar{c}_i \vee \bar{a}_i \bar{b}_i c_i \vee a_i b_i c_i \\
 &= \bar{p}_i c_i \vee p_i \bar{c}_i = c_i \oplus p_i
 \end{aligned} \tag{3.4}$$

Hierin ist p_i das "Propagate-Bit", das darüber entscheidet ob ein einkommendes Übertragsbit c_i in das Übertragsbit c_{i+1} weitergereicht wird oder nicht, was dann der Fall ist, wenn genau ein Bit a_i oder $b_i=1$ ist, also die ExOr-Verknüpfung = 1 ist. Es gilt dann

$$\begin{aligned}
 p_i &= LSB[a_i + b_i] \\
 &= \bar{a}_i b_i \vee a_i \bar{b}_i = a_i \oplus b_i
 \end{aligned} \tag{3.5}$$

Der Volladdierer liefert ein in die nächst höhere Bitstelle weiterzugebendes Übertragsbit aus der logischen Verknüpfung

$$\begin{aligned}
 c_{i+1} &= a_i b_i \vee c_i p_i \\
 &= g_i \vee c_i p_i = a_i b_i \vee a_i c_i \vee b_i c_i
 \end{aligned} \tag{3.6}$$

denn ein Übertragsbit c_{i+1} entsteht genau dann, wenn sich dies aus dem "Generate"-Bit $g_i = a_i b_i$ unmittelbar ergibt oder wenn das "Propagate"-Bit die Weitergabe eines einkommenden Bits c_i anzeigt.

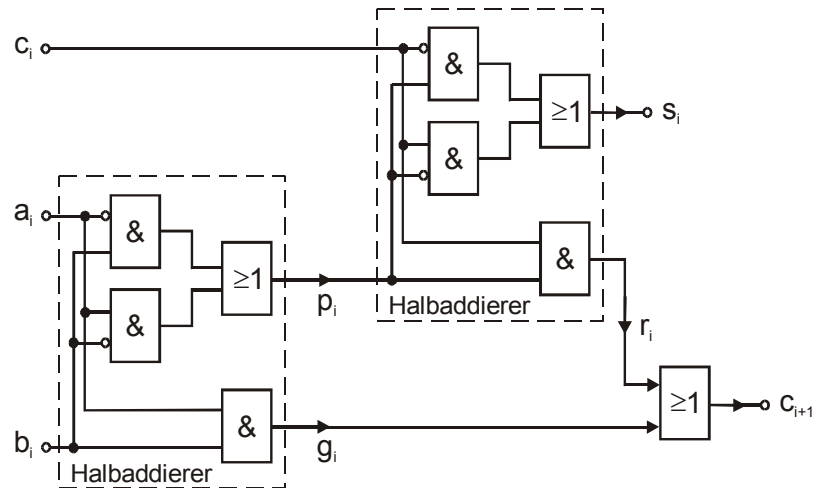


Bild 3.5: Halbaddierer-Realisierung eines Volladdierers

Bild 3.5 zeigt die klassische Schaltungsrealisierung auf der Basis von 2 Halbaddierern (siehe z.B. [SchiSchmi92]). Aus den Gleichungen (3.4), (3.6) kann auch eine schaltungstechnisch einheitlichere Lösung auf der Basis z.B. von NAND Gattern entwickelt werden, die man durch arithmetische Umwandlung der Booleschen Gleichung mithilfe des Satzes von de Morgan [SchiSchmi92] erhält.

Für das Übertragsbit c_{i+1} läßt sich noch eine schaltungstechnisch wichtige Variante angeben, die zu einer Multiplexer-Realisierung¹ führt.

$$\begin{aligned} c_{i+1} &= a_i \bar{p}_i \vee c_i p_i \\ &= b_i \bar{p}_i \vee c_i p_i \end{aligned} \quad (3.7)$$

Dies wird plausibel aus der Überlegung, dass die lokale Übertragserzeugung $a_i b_i$ auch mit Hilfe des Propagatebits ausgedrückt werden kann,

$$a_i b_i = a_i \bar{p}_i = b_i \bar{p}_i \quad (3.8)$$

denn ist $a_i b_i = 1$, dann ist definitionsgemäß $\bar{p}_i = 1$, so daß eines der beiden Bits durch $\bar{p}_i$ ersetzt werden kann.²

¹ Aufwandgünstig auf der Basis von Transmission-Gate-Technologie (siehe Kapitel 5)

² Ein algebraischer Nachweis gelingt auch durch einfaches Ersetzen von $\bar{p}_i$ in (3.8).

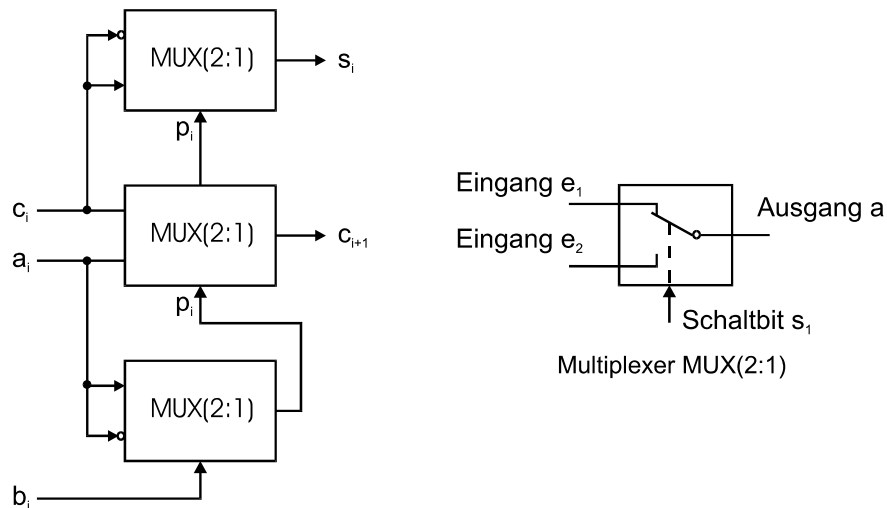


Bild 3.6: Multiplex(-Transmission-Gate)-Realisierung eines Volladdierers; Multiplexer MUX (2:1)

Derartige Multiplexer-Realisierungen sind technologisch von großer Bedeutung, da diese aufwandgünstig und funktionseffizient mithilfe von CMOS-Transmission-Gates (s. Kapitel 2.5) realisiert werden können. Im Falle des Volladdierers ergibt sich eine Aufwandsreduktion (Transistoranzahl) um ca. den Faktor 2 und auch die für schnelle Schaltungen bedeutsame Verzögerungszeit ist in ähnlicher Größenordnung reduziert.

Aus N Volladdierern läßt sich ein N -Bit Addierer für Dualzahlen mit N Bit Wortlänge aufbauen (siehe Bild 3.7). Die Übertragsrechnung erfolgt durch Weitergabe des Übertragsbits an die nächst höhere Stelle. Über c_0 und c_{N+1} kann eine weitere Kaskadierung, z.B. mit Modulen auf Zahlenwortebene, vorgenommen werden.

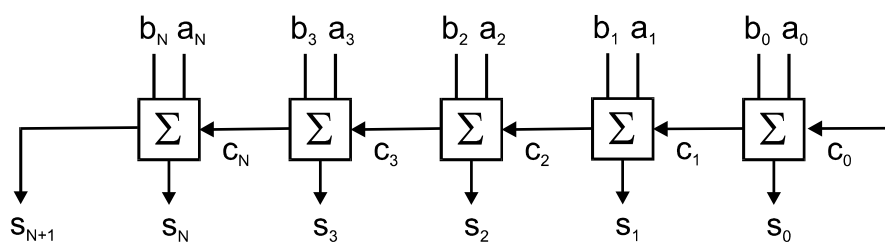


Bild 3.7: Addiererschaltung mit Übertragsweiterleitung

Typisch für diesen Addierertyp ist, daß die Volladdierer hintereinandergeschaltet sind und bezüglich des Übertrages alle Addierer durchlaufen werden müssen bis ein Ergebnis vollständig vorliegt (Ripple-Carry). Damit wird die Zeitdauer einer Addition durch die Laufzeit des Übertragbits bestimmt und die Additionszeit steigt proportional mit der Stufenzahl.

3.3.2 Addiererschaltnetz mit Übertragsvorausberechnung (Carry-Look-Ahead=CLA)

Es lässt sich eine zeitgünstigere Alternative zur Übertragsberechnung entwickeln, die zeitlich parallel zur Addition abläuft (siehe z.B. [WasFly 82]). Man geht hierzu von Gl. 3.6 aus:

$$c_{i+1} = g_i \vee (p_i c_i) = a_i b_i \vee c_i (\bar{a}_i b_i \vee a_i \bar{b}_i) \quad (3.9)$$

mit den intern erzeugten "Generate"- und "Propagate"-Variablen g_i und p_i . Mit g_i wird der Übertrag aufgrund der Eingangsvariablen a_i , b_i der jeweiligen Addiererzelle erzeugt, mit p_i das ankommende Übertragsbit weitergegeben oder absorbiert. Durch sukzessives Einsetzen erhält man

$$\begin{aligned} c_1 &= g_0 \vee (p_0 c_0) \\ c_2 &= g_1 \vee (p_1 c_1) = g_1 \vee (p_1 g_0) \vee (p_1 p_0 c_0) \\ c_3 &= g_2 \vee (p_2 c_2) = g_2 \vee (p_2 g_1) \vee (p_2 p_1 c_1) \\ &= g_2 \vee (p_2 g_1) \vee (p_2 p_1 g_0) \vee (p_2 p_1 p_0 c_0) \\ &\vdots \end{aligned} \quad (3.10)$$

d.h. Ausdrücke zur Berechnung der jeweiligen Überträge, die zwar immer komplizierter werden, aber dennoch eine parallele Berechnung in wenigen Gatterlaufzeiten gestatten, denn "Generate"- und "Propagate"-Bits entstehen direkt aus a_i und b_i . Die schaltungstechnische Realisierung zeigt Bild 3.8. Der "Carry-Look-Ahead"-Generator erhält den Eingangsübertrag c_0 , die "Generate"-Bits g_i und die "Propagate"-Bits p_i . Er erzeugt die Zwischenüberträge c_i zeitlich parallel zur Addition.

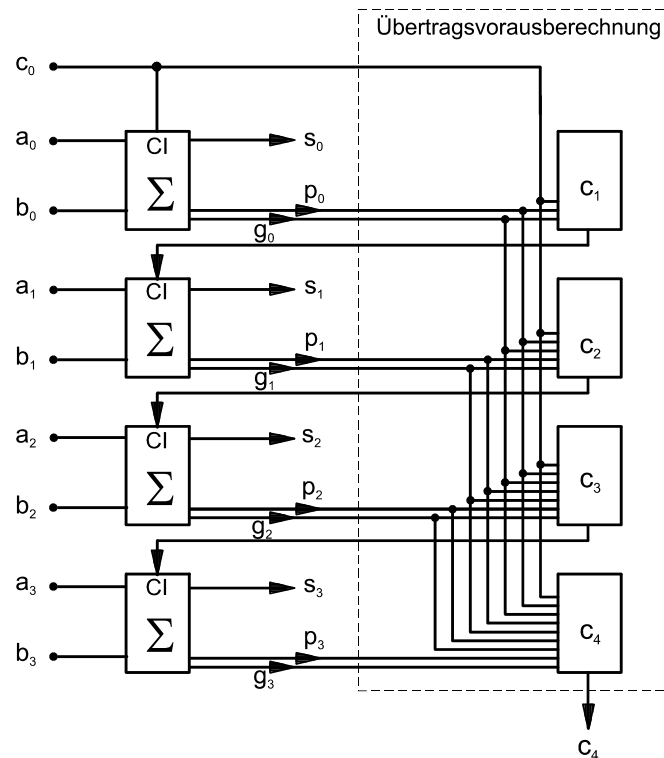


Bild 3.8: Addierschaltnetz mit Übertragsvorausberechnung

Auch hier läßt sich eine Realisierung auf der Basis von NAND-Gattern angeben. Ersichtlich wächst der Aufwand an nötigen Transistoren exponentiell mit der Wortlänge, so daß speziell optimierte CMOS-VLSI-Zellen auch hier sinnvoll sind und den Aufwand um eine Größenordnung verringern können [Pirsch96].

Der Zeitgewinn läßt sich folgendermaßen abschätzen: Beträgt der Volladdiererzeitbedarf z.B. 4 Gatterlaufzeiten ($4t_g$), so benötigt eine 4 Bit Addition $16t_g$ nach der Ripple-Carry-Methode.

Demgegenüber kann eine 4 Bit Addition nach dem Carry-Look-Ahead-Prinzip in

- $2t_g$ die "Generate"- und "Propagate"- Bits berechnen
- in weiteren $2t_g$ die Überträge im voraus berechnen
- in weiteren $2t_g$ die Summenbits berechnen,

so dass insgesamt in $6t_g$ eine Addition durchgeführt sein kann (im Sinne der dargestellten Schaltung).

Für eine weitere Kaskadierung ist eine Übertragsvorausberechnung wegen des exponentiellen Anstiegs des Aufwandes mit der Stellenzahl nicht direkt zweckmäßig. Man geht dann über zu einer hierarchischen Übertragsvorausberechnung bzw. wendet dann auf höherer Ebene das Konzept der Übertragsweiterleitung an. Das folgende Bild zeigt eine Kaskadierung von vier 4-Bit-Addierern mit

Übertragsvorausberechnung in den vier 4-Bit-Addierern, jeweils 4 Bit zusammenfassend, und Übertragsweiterleitung auf höherer Ebene.

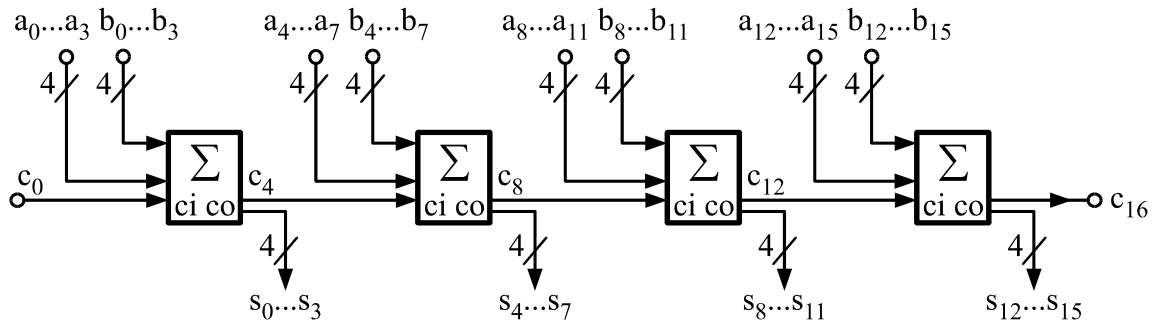


Bild 3.9: Addierschaltnetz mit Übertragsweiterleitung zwischen 4-Bit-Addiermodulen

Demgegenüber ist z. B. bei einem 16-Bit-Addierer eine Übertragsvorausberechnung in zwei hierarchisch getrennten Ebenen entwickelbar. Aus der Gleichung für den Übertrag c_{i+1} entnimmt man für c_4 :

$$c_4 = \underbrace{g_3 \vee (p_3 \wedge g_2) \vee (p_3 \wedge p_2 \wedge g_1) \vee (p_3 \wedge p_2 \wedge p_1 \wedge g_0)}_{g_3^1} \vee \left(\underbrace{p_3 \wedge p_2 \wedge p_1 \wedge p_0}_{p_3^1} \wedge c_0 \right)$$

$$c_4 = g_3^1 \vee p_3^1 c_0$$

eine Form, die der Berechnung von c_1 in der unteren Hierarchiestufe entspricht.

Ebenso läßt sich

$$\begin{aligned} c_8 &= g_7^1 \vee p_7^1 c_4 = g_7^1 \vee p_7^1 (g_3^1 \vee p_3^1 c_0) \\ &= g_7^1 \vee p_7^1 g_3^1 \vee p_7^1 p_3^1 c_0 \end{aligned}$$

berechnen.

Ersichtlich haben die Übertragsberechnungen in der zweiten Ebene dieselbe Form wie in der unteren, d. h. es läßt sich eine zweite Übertragsvorausberechnungsebene entwickeln, bei der mit denselben Logikschaltnetzen wie bereits beschrieben, die Addition auch für größere binäre Wortlängen mit "carry look ahead" berechnet werden kann. Das folgende Bild zeigt das zugehörige Schaltschema. Auf dieser Basis ist eine mehrstufige hierarchische Kaskadierung möglich.

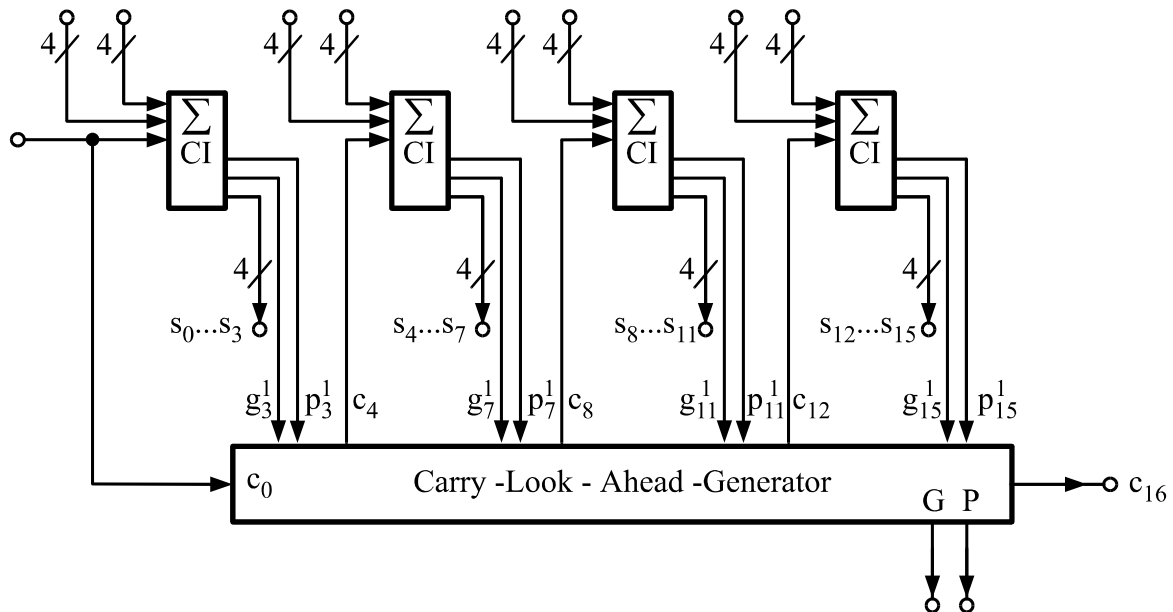


Bild 3.10: Addierschaltnetz mit hierarchischer Übertragsvorausberechnung

3.3.3 Akkumulation von Dualzahlen

Häufig ist in der digitalen Signalverarbeitung eine fortlaufende Addition von Dualzahlen (Akkumulation) erforderlich, z. B. bei einer Faltung oder Korrelation, aber auch bei der Abarbeitung eines Multiplikationsalgorithmus mit fortlaufender Addition der Teilprodukte. Dies kann prinzipiell in serieller Technik mithilfe eines Rechenwerkes, bestehend aus einem Addierschaltnetzwerk mit einer Wortlänge, die für die Gesamtsumme erforderlich ist, und einem entsprechenden Akkumulationsregister R erfolgen (siehe Bild 3.11).

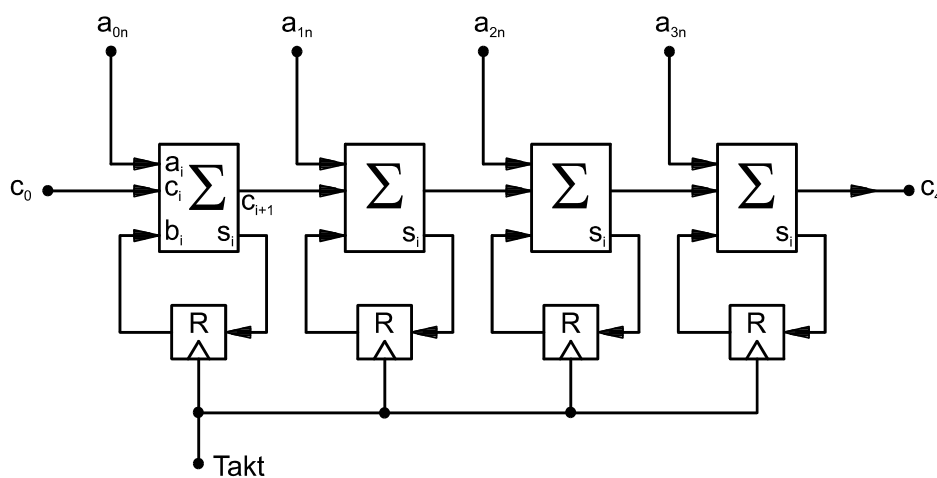


Bild 3.11: Serielle Akkumulation

Mit jedem Takt wird eine neue Zwischensumme ins Register übernommen und es kann eine neue Dualzahl $a_{0n}, a_{1n}, a_{2n}, \dots$ addiert werden. Angenommen ist im obigen

Bild eine einfache Übertragsweiterleitung. In vielen Fällen ist die serielle Akkumulation aus zeitlichen Gründen oder Architektur-Gesichtspunkten nicht möglich und es muß eine parallele z.B. paarweise Zusammenfassung entsprechend Bild 3.12 erfolgen.

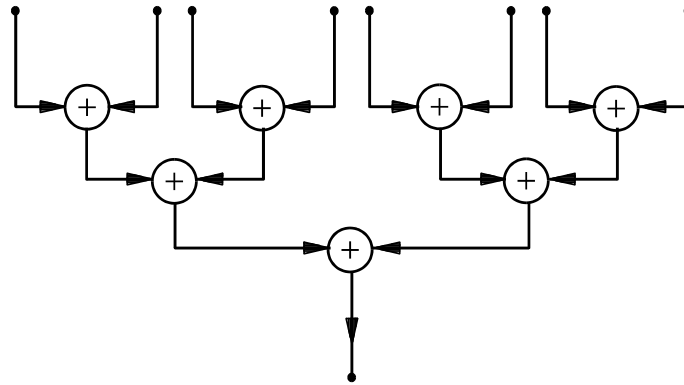


Bild 3.12: Parallele Akkumulation

Die Addierer in dieser Schaltung können dabei z.B. mit Übertragsweiterleitung oder Übertragsvorausberechnung realisiert sein.

In zeitkritischen Fällen erweist sich dabei das "Carry-Save"-Konzept (ausführliche Beschreibung z.B. in [WasFly 82]), ein Verfahren mit Übertragsakkumulation, als besonders günstig. Bild 3.13 zeigt zunächst das Prinzip des "Carry-Save"-Addierers.

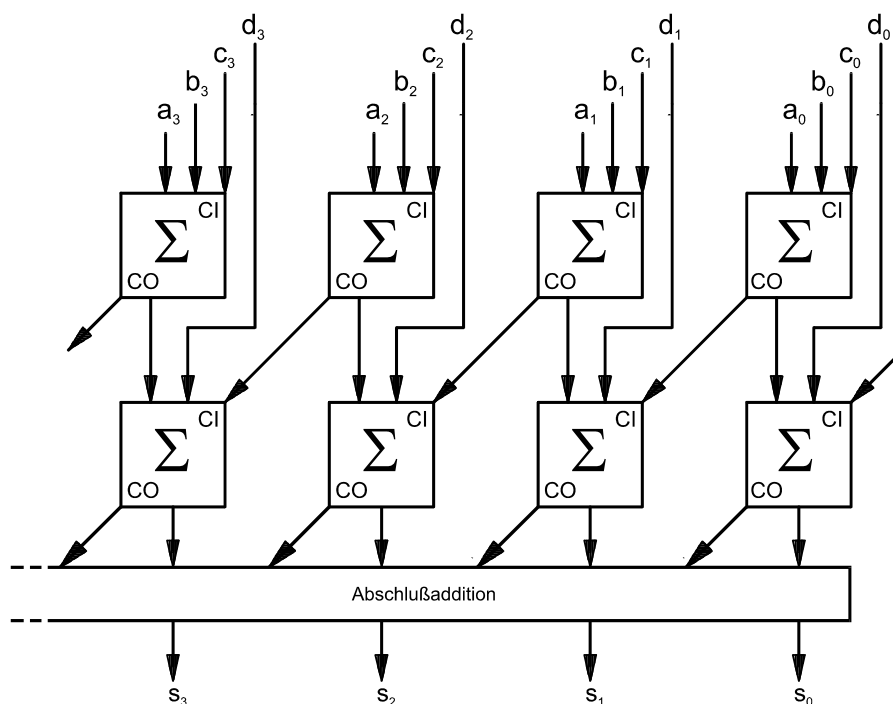


Bild 3.13: Zweistufiger "Carry-Save"-Addierer aus Volladdierern

Sind zum Beispiel (Bild 3.13) vier Zahlen $\{a_3, a_2, a_1, a_0\}$, $\{b_3, b_2, b_1, b_0\}$, $\{c_3, c_2, c_1, c_0\}$, $\{d_3, d_2, d_1, d_0\}$ zu addieren, so können in der ersten Stufe unter Nutzung des Carry-Eingangs drei Zahlen addiert werden. Der entstehende Übertrag wird an die nächste Stufe weitergegeben, in der die Summe aus der ersten Stufe und der vierte Summand addiert werden. Die Laufzeit der einzelnen Addierstufen aus N Volladdierern ist unabhängig von der Stellenzahl, weil die Überträge in die nächste Stufe weitergegeben werden. Stets muß bei der Anwendung des Carry-Save-Prinzips eine Abschlußaddition („Vector merging adder“) erfolgen, um die Übertragsbits der letzten Stufe mit den Summenbits zu verrechnen. Diese Addition erfolgt z.B. mithilfe der Übertragsweiterleitung oder -vorausberechnung. Das "Carry-Save"-Prinzip läßt sich vorteilhaft auch für ein Addierfeld verwenden (Bild 3.14).

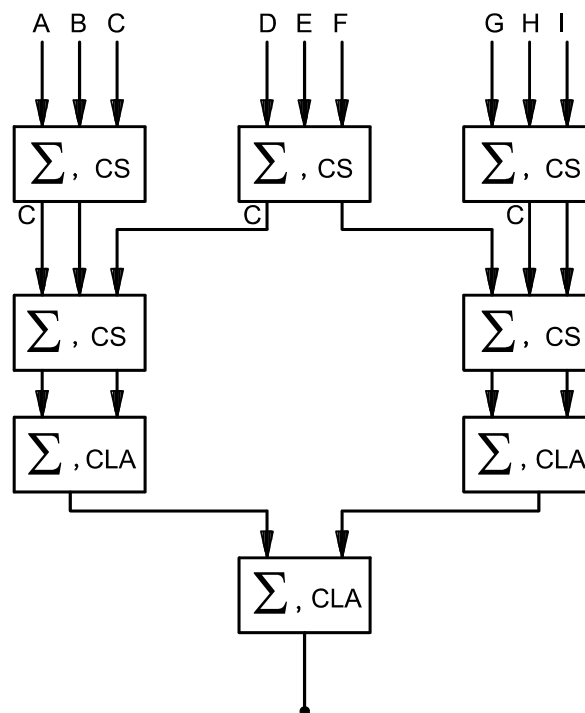


Bild 3.14: Additionsbaum zur Akkumulation von 9 Summanden, CS: Carry-Save-, CLA: Carry-Look-Ahead-Prinzip

Hier werden vom Carry-Save-Addierer einer Stufe stets aus drei Summanden zwei Teilsummen gebildet (eine ist die Übertragssumme). In der nächsten Stufe wird dann entsprechend weiter zusammengefaßt. Ab vier Summanden abwärts macht das Carry-Save-Prinzip aus der Sicht der Additions geschwindigkeit keinen Sinn mehr, da dann eine zusätzliche Addiererstufe entsteht. Stattdessen wird z. B. zweckmäßig in

zwei CLA-Stufen die Akkumulation zu Ende geföhrt. Besondere Bedeutung hat das Carry-Save-Prinzip bei schnellen Array-Multiplizierern³.

3.3.4 Rechenwerk zur Addition und Subtraktion von Zweierkomplementzahlen

Für eine Zweierkomplementzahl A mit $N+1$ Stellen inklusive Vorzeichen war definiert worden

$$A = -a_N \cdot 2^N + \sum_{i=0}^{N-1} a_i \cdot 2^i \quad (3.11)$$

Wird eine zweite Zahl B dazu addiert, so ergibt sich

$$S = A + B = (-a_N - b_N) \cdot 2^N + \sum_{i=0}^{N-1} (a_i + b_i) \cdot 2^i \quad (3.12)$$

Diese Addition werde in einem $N+1$ stelligen Addierschaltnetz, bestehend aus $N+1$ Volladdierern und mithilfe der Übertragsweiterleitung (Ripple-Carry) ausgeführt.

Es entsteht dabei entsprechend Gl. (3.4) eine Zahl S mit dem dargestellten Summenwert

$$\begin{aligned} S &= (-a_N - b_N) \cdot 2^N + c_N \cdot 2^N + \sum_{i=0}^{N-1} s_i \cdot 2^i \\ &= (-a_N - b_N + c_N) \cdot 2^N + \sum_{i=0}^{N-1} s_i \cdot 2^i \\ &= -s_N \cdot 2^N + \sum_{i=0}^{N-1} s_i \cdot 2^i \end{aligned} \quad (3.13)$$

mit c_N dem Übertrag für die letzte Stelle und s_N dem Vorzeichenbit des als Zweierkomplementzahl dargestellten Ergebnisses.

³ Hier insbesondere auch in der Form des Wallace-Trees [Wall64], bei dem im zeitkritischen Pfad eine minimale Addiererzahl vorliegt.

Eingang			Ausgang		Überlaufanzeige	Fall
a_N	b_N	c_N	S_N	c_{N+1}	$\ddot{u}$	Nr.
0	0	0	0	0	0	I
0	0	1	1	0	1 (Überlauf)	II
0	1	0	1	0	0	III
1	0	0	1	0	0	IV
0	1	1	0	1	0	V
1	0	1	0	1	0	VI
1	1	0	0	1	1 (Überlauf)	VII
1	1	1	1	1	0	VIII

Tabelle 3.3: Wahrheitstabelle zur Zweierkomplement-Addition, Überlaufverhalten

Aus (3.13) können nun die Überlaufbedingungen abgelesen werden. Wenn nämlich die Summe S mit der auch für die Summanden verwendeten Wortlänge darstellbar sein soll, so muß $s_N \in \{0,1\}$ sein. Hieraus kann eine Wahrheitstabelle (Tabelle 3.3) zur Bestimmung von Vorzeichenbits und Überlauf durch Betrachtung des letzten Volladdierers entwickelt werden. Die Zweierkomplement-Addition kann dabei anschaulich am Zahlenkreis dargestellt werden. Die Addition positiver und negativer Zahlen (damit ist die Subtraktion durch Komplementierung eingeschlossen) wird dabei als Addition der Dualzahlen über eine Addition von Kreissegmenten symbolisiert. Dabei werden nur positive oder gemischt positive und negative oder nur negative Zahlen addiert. Je nachdem, ob dabei positive oder negative Zahlen entstehen kann dabei ein Überlauf festgestellt werden. Anhand von 2 Zahlenbeispielen entsprechend den markierten Zeilen in Tabelle 3.3 kann nochmals die zyklische Eigenschaft des Zweierkomplements in Verbindung mit der Addition studiert werden. Ein Übertragsbit zur $N+1$ ten Stelle tritt in den Fällen V, VI und VIII auf, ohne daß Überlauf entsteht. Überlauf entsteht in den Fällen II und VII. Dies ist sichtbar daran, daß im Vorzeichenbit unsinnige Ergebnisse erscheinen

An einzelnen Fallbeispielen sollen die verschiedenen Möglichkeiten verdeutlicht werden.

Fall Nr. Dezimaladdition Binäraddition

I	$\begin{array}{r} 2 \\ + 3 \\ \hline 5 \end{array}$	$\begin{array}{rcccccl} (N+1) & N & 2 & 1 & 0 & i \\ & 0. & 0 & 1 & 0 & a_i \\ & 0. & 0 & 1 & 1 & b_i \\ & & & 1 & & c_i \\ \hline & 0. & 1 & 0 & 1 & s_i \end{array}$	
II	$\begin{array}{r} 7 \\ + 6 \\ \hline 13 \end{array}$	 $\begin{array}{rcccccl} & 0 & 1 & 1 & 1 & a_i \\ & 0. & 1 & 1 & 0 & b_i \\ & & 1 & 1 & & c_i \\ \hline & 1. & 1 & 0 & 1 & s_i \end{array}$ 	
V/VI	$\begin{array}{r} 3 \\ + (-1) \\ \hline 2 \end{array}$	$\begin{array}{rcccccl} & 0. & 0 & 1 & 1 & a_i \\ & 1. & 1 & 1 & 1 & b_i \\ & 1 & 1 & 1 & 1 & c_i \\ \hline \nearrow & 0. & 0 & 1 & 0 & s_i \end{array}$	
III/IV	$\begin{array}{r} 1 \\ + (-7) \\ \hline -6 \end{array}$	$\begin{array}{rcccccl} & 0. & 0 & 0 & 1 & a_i \\ & 1. & 0 & 0 & 1 & b_i \\ & & & 1 & & c_i \\ \hline & 1. & 0 & 1 & 0 & s_i \end{array}$	

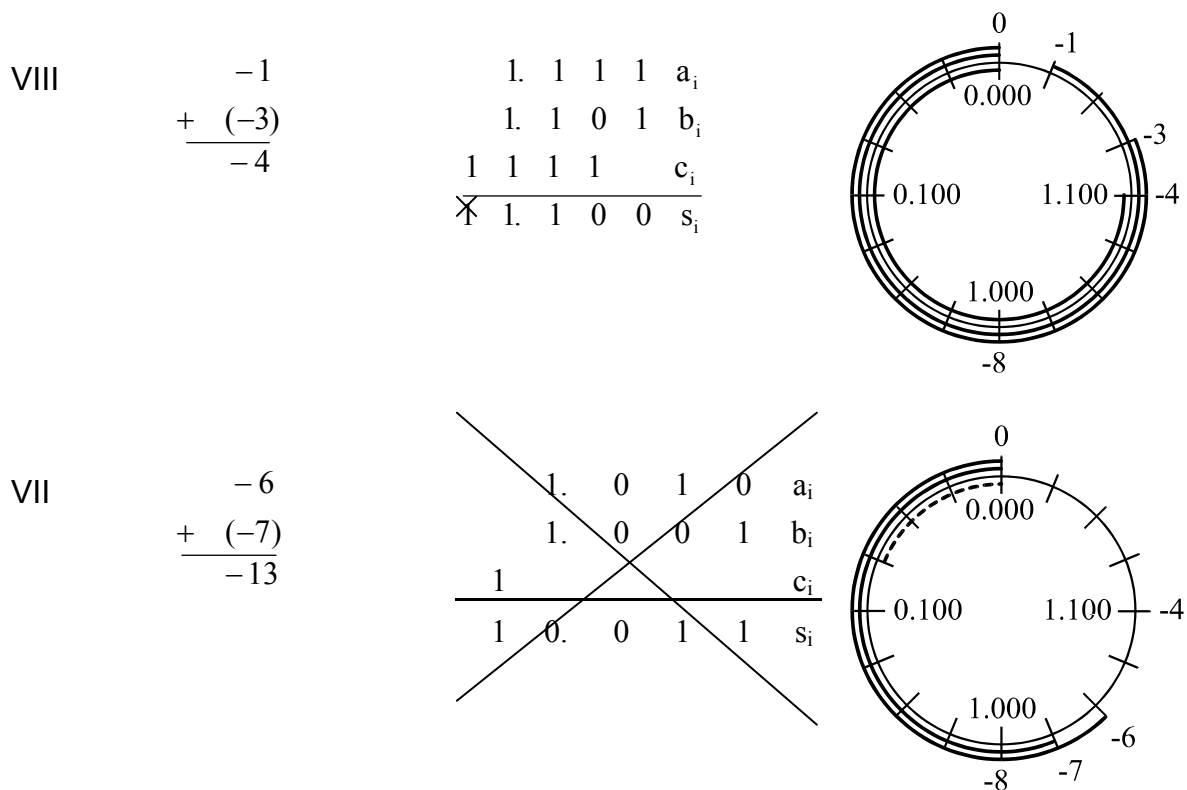


Bild 3.15: Fallbeispiele zur Zweierkomplement-Addition

Insbesondere kann an diesen Beispielen noch einmal die zyklische Eigenschaft des Zweierkomplements in Verbindung mit der Addition studiert werden. Ein Übertragsbit zur (N+1)-ten Stelle tritt auf in den Fällen V, VI, VIII ohne daß Überlauf entsteht.

Überlauf tritt auf in den Fällen II und VII. Dies ist dann gegeben, wenn im Vorzeichenbit unsinnige Ergebnisse auftreten.

Die Überlaufanzeige kann aus der Wahrheitstabelle abgeleitet werden. Es müssen die Vorzeichenbit gleich sein ($a_N \equiv b_N$) und Übertragbit c_{N+1} und Summenbit s_N ungleich ($\neq$) sein:

$$\ddot{u} = (a_N \equiv b_N) \wedge (c_{N+1} \neq s_N)$$

Ist in einem Addierwerk auch das Übertragbit c_N verfügbar, so kann vereinfacht

$$\ddot{u} = c_{N+1} \neq c_N$$

verwendet werden. Im übrigen kann an den Zahlenringdarstellungen abgelesen werden, daß bei Überlauf die entstandene Zahl zwar nicht weiter in diesem N+1-stelligen Rechenwerk verwendet werden kann. Sie kann aber als eine N+2-stellige Zahl im 2er Komplement verwendet werden in Verbindung mit einer Zahlenbereichserweiterung!

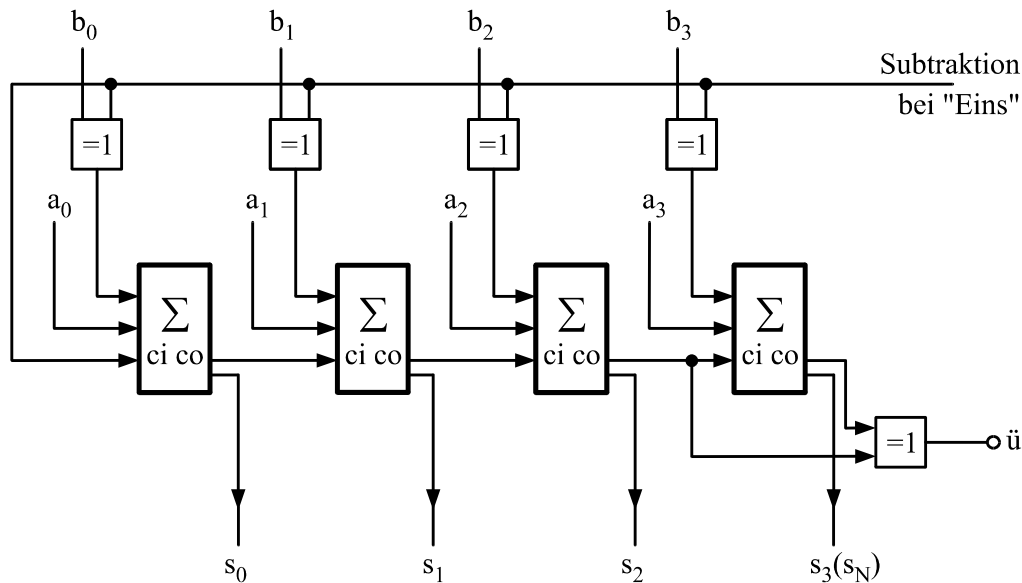


Bild 3.16: Addiererschaltnetz zur Addition und Subtraktion von 2er Komplement-Zahlen

3.3.5 Arithmetisch-logische Einheiten

Arithmetisch-logische Einheiten sind in der Lage, einstellbar weitgehend beliebige arithmetische und logische Verknüpfungen der Eingangsworte vorzunehmen. Solche "ALU's" sind Teil eines Prozessors bzw. Rechenwerks. Übliche Operationen sind z.B. verschiedene Formen der Addition und Subtraktion für unterschiedliche Zahlenformate wie auch logische Verknüpfungen wie UND, ODER, Negation etc.

Bild 3.17 zeigt in Verbindung mit Tabelle 3.4 als Beispiel eine 4-Bit-ALU, die mit 5 Steuerbits 32 verschiedene Verknüpfungen $z = x \Psi y$ ausführen kann und mit CI, CO, CG, CP für Übertragsvorausberechnung geeignet ist. (CI, CO: Carry In, Carry Out; CG, CP: Carry Generate, Carry Propagate zur hierarchischen Carry-Look-Ahead-Kontrolle)

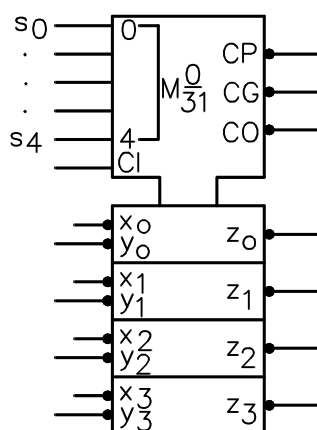


Bild 3.17: 4-Bit ALU

Steuergrößen				logische Operationen	arithmetische Operationen $s_4 = 0$	
s_0	s_1	s_2	s_3	$s_4 = 1$	Übertrag $u_0 = 0$	Übertrag $u_0 = 1$
0	0	0	0	$z = 0$	$z = -1$	$z = 0$
0	0	0	1	$z = x \vee y$	$z = x \vee y$	$z = (x \vee y) + 1$
0	0	1	0	$z = \bar{x} \wedge y$	$z = x \vee \bar{y}$	$z = (x \vee \bar{y}) + 1$
0	0	1	1	$z = \bar{x}$	$z = x$	$z = x + 1$
0	1	0	0	$z = x \wedge y$	$z = x \wedge \bar{y} - 1$	$z = x \wedge \bar{y}$
0	1	0	1	$z = \bar{y}$	$z = (x \vee y) + (x \wedge \bar{y})$	$z = (x \vee y) + (x \wedge \bar{y}) + 1$
0	1	1	0	$z = x \neq y$	$z = x - y - 1$	$z = x - y$
0	1	1	1	$z = x \wedge \bar{y}$	$z = (x + \bar{y}) + x$	$z = (x \wedge \bar{y}) + x + 1$
1	0	0	0	$z = x \wedge y$	$z = (x \wedge y) - 1$	$z = x \wedge y$
1	0	0	1	$z = x \equiv y$	$z = x + y$	$z = x + y + 1$
1	0	1	0	$z = y$	$z = (x \vee \bar{y}) + (x \wedge y)$	$z = (x \vee \bar{y}) + (x \wedge y) + 1$
1	0	1	1	$z = \bar{x} \vee y$	$z = (x \wedge y) + x$	$z = (x \wedge y) + x + 1$
1	1	0	0	$z = x$	$z = x - 1$	$z = x$
1	1	0	1	$z = x \vee \bar{y}$	$z = (x \vee y) + x$	$z = (x \vee y) + x + 1$
1	1	1	0	$z = x \vee y$	$z = (x \vee \bar{y}) + x$	$z = (x \vee \bar{y}) + x + 1$
1	1	1	1	$z = 1$	$z = x \cdot 2$	$z = x \cdot 2 + 1$

Tabelle 3.4: Funktionstabelle einer Arithmetisch-Logischen Einheit

Die technische Realisierung einer ALU erfolgt dabei z.B. analog zur Addierer-Schaltkette mit Ripple-Carry als "Bitslice"-Schaltkette entsprechend Bild 3.18 [MeadCon80],[Pirsch86].

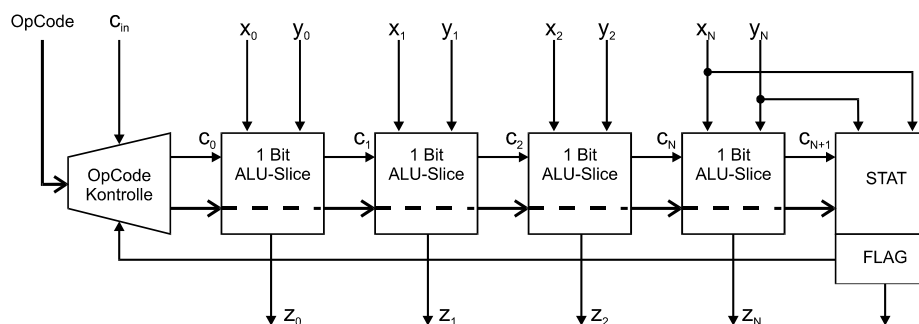


Bild 3.18: Arithmetisch-Logische Einheit (ALU)

Prinzipiell kann dabei die arithmetische oder logische Funktion der ALU mit Universallogikbausteinen nach Bild 3.19 erzielt werden.

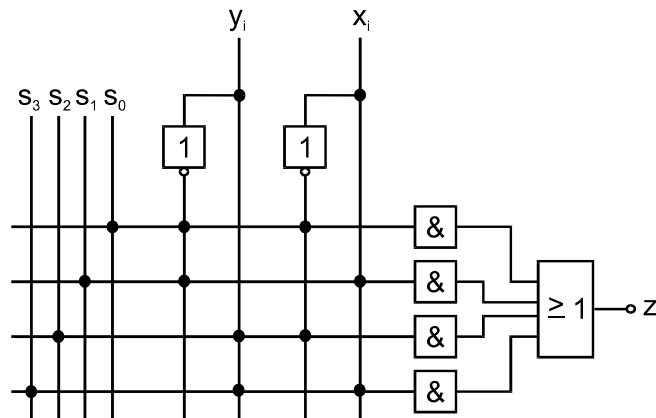


Bild 3.19: Universal-Logikbaustein (ULB)

Hierzu wird eine logische Funktion bestehend aus den 4 Mintermen der Variablen x_i , y_i realisiert, die über die Steuervariablen $s_0 \dots s_3$ ausgewählt werden können

$$z_i = s_0 \bar{x}_i \bar{y}_i \vee s_1 x_i \bar{y}_i \vee s_2 \bar{x}_i y_i \vee s_3 x_i y_i \quad (3.14)$$

Damit lassen sich über das Steuerwerk $\{s_0, s_1, s_2, s_3\}$ alle 16 möglichen zweistelligen Booleschen Verknüpfungen der Variablen x_i , y_i programmieren. Die Vielfalt dieser Verknüpfungen ist in Bild 3.19 enthalten. Die arithmetischen Verknüpfungen der Variablen sind dabei auf ihre Booleschen Äquivalente abzubilden.

Neben der Verknüpfungen der Operandenbits ist jeweils auch ein Übertragsbit zu erzeugen, so daß insgesamt [MeadCon80]

- 3 Universal-Logikbausteine (ULB) nach Bild 3.19
 - 2 ULB für das Ergebnisbit z_i unter Berücksichtigung des Carrybits,
 - 1 ULB zur Bildung des neuen Carrybits und
- eine zusätzliche Übertragslogik

benötigt werden, um eine 1Bit ALU-Slice nach Bild 3.18 aufzubauen.

3.4 Schaltwerke zur binären Multiplikation

Die binäre Multiplikation ist insbesondere für Aufgaben der Signalverarbeitung von besonderer Bedeutung. Eine Vielzahl von Algorithmen (Faltung, Korrelation, Spektraltransformationen) lassen sich in der Form $\sum_i a_i x_i$ also als Akkumulation von

Teilprodukten darstellen. Zugleich sind Multiplizierer-Module auch in vielen Rechnerbausteinen enthalten. Daher sollen Grundstrukturen der Multiplikation hier behandelt werden.

3.4.1 Seriell-parallele Multiplikation von Dualzahlen in Betrag-Vorzeichen-Darstellung

Offensichtlich ist die Darstellung einer Dualzahl nach Betrag und Vorzeichen für eine Multiplikation besonders günstig, da eine getrennte Verarbeitung nach Betrag und Vorzeichen einfach möglich ist. Es gilt nach für eine $N+1$ stellige Dualzahl

$$A = (-1)^{a_N} \cdot \sum_{i=0}^{N-1} a_i \cdot 2^i = (-1)^{a_N} \cdot |A| \quad (3.15)$$

a_N : Vorzeichenbit (= 0 für positive Zahlen)

Wird das Produkt zweier Dualzahlen $A \cdot B$ gebildet, so gilt mit entsprechender Definition auch für B für dieses Produkt

$$\begin{aligned} P = A \cdot B &= (-1)^{b_N} \cdot (-1)^{a_N} \cdot |A| \cdot |B| = (-1)^{a_N + b_N} \cdot |A| \cdot |B| \\ &= (-1)^{p_N} \cdot |P| \end{aligned} \quad (3.16)$$

Das *Vorzeichen des Produktes* kann einfach über folgende *Wahrheitstabelle* ermittelt werden:

a_N	b_N	$a_N + b_N$	p_N
0	0	0	0
0	1	1	1
1	0	1	1
1	1	2	0

und es gilt für die Vorzeichenbildung des Produktes die ExOr-Beziehung

$$p_N = a_N \oplus b_N \quad (3.17)$$

Entsprechend ergibt sich für den Betrag des Produktes

$$\begin{aligned} |P| &= |A| \cdot |B| = |A| \cdot \sum_{i=0}^{N-1} b_i \cdot 2^i \\ &= |A| \cdot b_0 + |A| \cdot b_1 \cdot 2 + |A| \cdot b_2 \cdot 2^2 + \dots + |A| \cdot b_{N-1} \cdot 2^{N-1} \end{aligned} \quad (3.18)$$

Arbeitet man die angegebene Gleichung ab, z.B. beginnend mit dem MSB b_{N-1} , so erhält man die am folgenden Beispiel dargestellte Prozedur

		b_{N-1}	$\dots$	b_1	b_0						
		.		.	.						
		.		.	.						
		.		.	.						
$ A \cdot B :$	1	0	1	1	.	1	1	0	1		
$ A \cdot b_{N-1} \cdot 2^{N-1}$			1	0	1		1	0	0	0	
+ $ A \cdot b_{N-2} \cdot 2^{N-2}$	+			1	0		1	1	0	0	(3-1)
	1	0	0	0		0	1	0	0		
+ $ A \cdot b_1 \cdot 2$	+				0		0	0	0	0	
	1	0	0	0		0	1	0	0		
+ $ A \cdot b_0$	+					1	0	1	1		
	1	0	0	0		1	1	1	1		

Dieses elementare Rechenschema ist Grundlage für die technische Realisierung der Multiplikation in elementaren Rechenwerken oder Assembler-Programmen, die nach dem "Shift and Add"-Prinzip arbeiten. Die serielle Ausführung des Rechenschemas kann z.B. in einem Rechenwerk mit einem geeigneten Steuerprogramm (Mikroprogramm) durchgeführt werden (Bild 3.20).

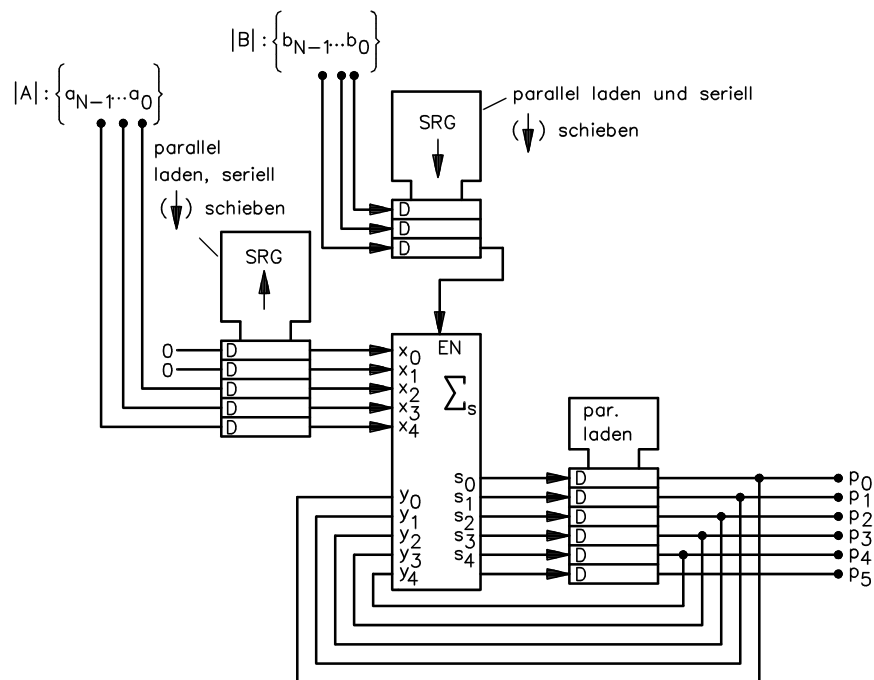


Bild 3.20: Seriell-parallele Multiplikation von Dualzahlbeträgen, Beginn mit MSB⁴

Die Multiplikation mit dem einzelnen Bit $(|A| \cdot b_i \cdot 2^i)$ wird dabei wortweise, d.h. bezogen auf $|A|$ (bit-) parallel ausgeführt. Man bezeichnet dieses Verfahren daher als seriell-parallele Multiplikation. Von Stufe zu Stufe ist dabei $|A|$ mit dem jeweiligen Bit und dessen Wertigkeit gewichtet zu addieren. Dies wird durch serielles Schieben in den Schieberegistern (SRG) und durch über den Enable (EN)-Eingang des Summeriers gesteuerte Addition in Verbindung mit einer seriellen Akkumulation mit dem nachgeschalteten Akkumulation-SRG erreicht.

Mit dem dargestellten Rechenwerk kann das besprochene Rechenschema abgearbeitet werden. $|A|$ und $|B|$ werden jeweils in D-Schieberegister geladen, die auch in der Lage sind, seriel aufwärts ($|A|$) bzw. abwärts ($|B|$) zu schieben. Schrittweise wird zum Ausgangsschieberegisterinhalt $|A|$ oder 0 addiert. Dies kann von den einzelnen Bit b_i von $|B|$ über den Steuereingang EN des Addierers gesteuert werden. Bei jedem Schritt wird $|B|$ geschoben und - angefangen mit dem MSB b_{N-1} - ein neues Bit an den Steuereingang gelegt. Bei jedem Schritt wird auch $|A|$ in seiner Wertigkeit entsprechend dem Rechenschema geschoben.

⁴ Der Addierer ist hierbei schematisch mit einem Enable-Eingang (EN) ausgestattet, der eine Addition von $|A|$ gestattet oder statt $|A|$ nur Null addiert

3.4.2 Paralleles Multiplikationswerk

Die Dualzahlenmultiplikation kann auch auf eine Polynommultiplikation zurückgeführt werden:

$$\begin{aligned}
 |P| &= |A| \cdot |B| = \sum_{i=0}^{N-1} a_i \cdot 2^i \cdot \sum_{j=0}^{N-1} b_j \cdot 2^j \\
 &= \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} a_i \cdot b_j \cdot 2^{i+j} = \sum_{k=0}^{2N-2} p_k \cdot 2^k = \sum_{k=0}^{2N-2} q_k \cdot 2^k
 \end{aligned}$$

Die Koeffizienten q_k lassen sich durch eine Matrix-Vektormultiplikation ermitteln. Dabei entstehen die Koeffizienten q_k aus der unmittelbaren Addition der Koeffizientenprodukte $(a_i \cdot b_j)$ für gleiche Wertigkeiten $(i+j)$ und hieraus die Koeffizienten p_k (binäre Stellen) durch Addition und Berücksichtigung der Überträge.

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ \vdots \\ q_{N-1} \\ q_N \\ q_{N+1} \\ \vdots \\ q_{2N-2} \end{bmatrix} = \begin{bmatrix} a_0 & 0 & 0 & \dots & 0 \\ a_1 & a_0 & 0 & \dots & 0 \\ a_2 & a_1 & a_0 & \dots & 0 \\ \vdots & \vdots & a_2 & a_1 & \dots & 0 \\ \vdots & \vdots & \vdots & a_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \dots & 0 \\ a_{N-1} & \vdots & \vdots & \vdots & \dots & a_0 \\ \vdots & a_{N-1} & \vdots & \vdots & \dots & a_1 \\ \vdots & \vdots & a_{N-1} & \vdots & \dots & a_2 \\ 0 & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \vdots & \dots & a_{N-1} \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \vdots \\ \vdots \\ b_{N-1} \end{bmatrix}$$

Für eine Multiplikation zweier 3 Bit langer Dualzahlenbeträge als Beispiel ist im folgenden das Rechenschema noch näher erläutert:

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} a_0 & 0 & 0 \\ a_1 & a_0 & 0 \\ a_2 & a_1 & a_0 \\ 0 & a_2 & a_1 \\ 0 & 0 & a_2 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

hieraus folgt

$$|p| = a_0 \cdot b_0 + (a_1 \cdot b_0 + a_0 \cdot b_1) \cdot 2^1 + (a_2 \cdot b_0 + a_1 \cdot b_1 + a_0 \cdot b_2) \cdot 2^2 \\ + (a_2 \cdot b_1 + a_1 \cdot b_2) \cdot 2^3 + a_2 \cdot b_2 \cdot 2^4$$

Aus dem Schema einer systematischen Abarbeitung des Matrix-Vektorproduktes läßt sich leicht ein paralleles Multiplikationswerk entwickeln, wenn man die Teilprodukte gleicher Wertigkeit $i+j$ in vertikal untereinander angeordneten Zellen (um 90° gedrehte Matrix) bildet und addiert.

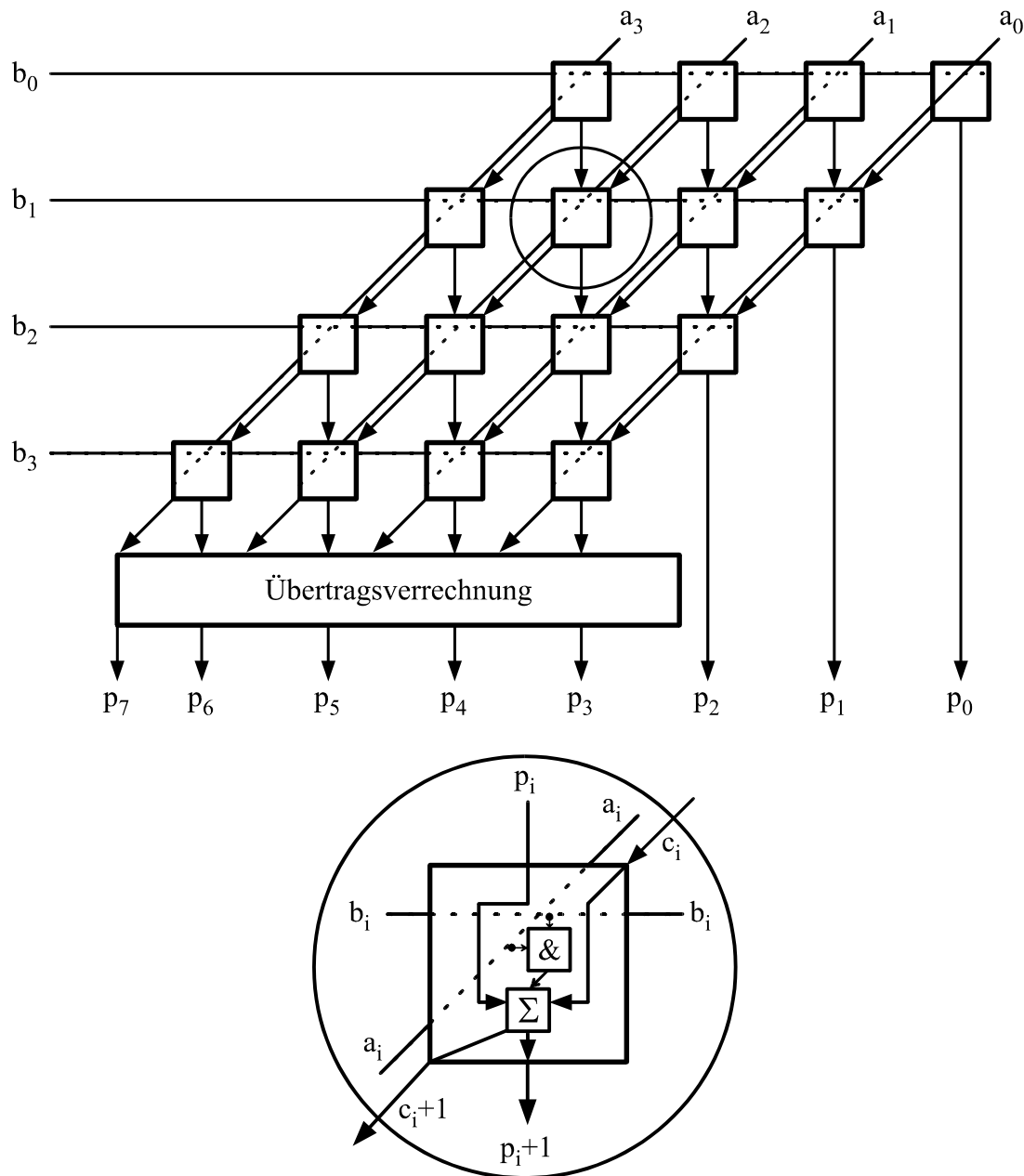


Bild 3.21: Paralleles Multiplikationsfeld, Multiplizierierzelle

Jede einzelne Zelle erhält von der vertikal darüberliegenden das jeweilige Teilprodukt p_i und von der diagonal rechts liegenden den Übertrag c_i aus der nächstniedrigen Wertigkeit. Die Bits a_i und b_i der Faktoren werden von Zelle zu Zelle horizontal bzw. vertikal durchgeschleift. Jede Zelle enthält einen Volladdierer und ein UND-Gatter und liefert ein Teilprodukt p_{i+1} und einen Übertrag c_{i+1} , der in sogenannter "Carry-Save" Technik diagonal weitergegeben wird. Selbstverständlich besteht die Möglichkeit, durch Pipelineregister (z. B. vertikal) den Datendurchsatz zu erhöhen. Erwähnt werden muß noch, daß am Ende des Feldes eine geeignete Übertragsverrechnung die Produktbits korrekt erzeugen muß. Solche parallelen Multiplikationswerke zeichnen sich durch eine besonders hohe Rechengeschwindigkeit aus.

Kapitel 4

Schaltwerke

4.1 Begriff und Funktion des Schaltwerkes

Die bisher ausschließlich behandelten Schaltnetze sind geeignet, um kombinatorische Logikverknüpfungen vorzunehmen. Auf diese Weise können logische Operationen wie z. B. Addition, Codierung, Fehlererkennung und Multiplexer realisiert werden. Schaltnetze können hingegen, weil sie kombinatorisch arbeiten, keine Informationen speichern und somit auch keine sequentiellen Algorithmen abarbeiten.

Schaltwerke sind dagegen Logikschaltungen, die Datenwörter speichern können und deren Ausgangsvariablen nicht mehr nur kombinatorisch aus den aktuellen Eingangsvariablen entstehen. Die Ausgangsvariablen hängen vielmehr auch von Eingangsvariablen früherer Zeitpunkte ab und sind Ergebnis eines algorithmischen Ablaufes. Schaltwerke können somit Schaltzustände speichern und sequentielle Abläufe steuern.

Zur Realisierung von Schaltwerken werden neben den Schaltnetzen geeignete Speicherbausteine für binäre Variablen verwendet, die zunächst beschrieben werden. Hierauf aufbauend werden dann formale Beschreibungsformen entwickelt, die zur Darstellung, d. h. zur Analyse und Synthese von Schaltwerken, nützlich sind.

Das allgemeine Blockschaltbild eines Schaltwerkes entsprechend Bild 4.1

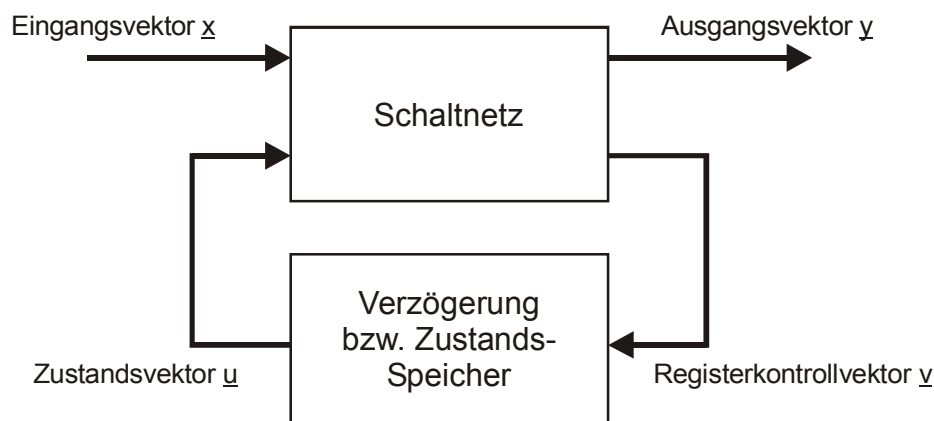


Bild 4.1: Blockschaltbild eines Schaltwerkes

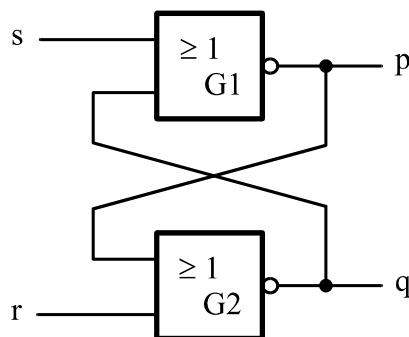
besteht prinzipiell aus einem rückgekoppelten Schaltnetz mit Verzögerungs- bzw. Speicherbausteinen als Zustandsspeicher in der Rückführung. Das hier gehaltene binäre Datenwort kennzeichnet den Zustand des Schaltwerkes. Ein Zustandswechsel, d. h. eine Änderung des Zustandsvektors $\underline{u}$, kann von einer Änderung des Eingangsvektors $\underline{x}$ und/oder dem Zustandsvektor selbst ausgelöst werden. Abhängig hiervon ergibt sich der jeweilige Ausgangsvektor $\underline{y}$.

4.2 Elementare Schaltwerke (Register und Flipflops)

4.2.1 SR-Flip Flop (SR-FF) zur Binärsignalspeicherung

Flip Flops sind Grundbausteine von Speichern, Zählern, Schieberegistern und Frequenzteilern. Ein Flip Flop besitzt zwei stabile Zustände und verharrt in einem aktuellen Zustand bis durch ein geeignetes Eingangssignal der andere stabile Zustand erreicht wird. Der jeweilige Zustand bzw. "Speicherinhalt" ist am Ausgang verfügbar. Es ist mit einem Flip Flop daher eine 1 Bit-Speicherung möglich.

Ein elementares SR-Flip Flop (SR-Latch) aufgebaut aus zwei NOR-Gattern zeigt Bild 4.2 (ein Aufbau aus NAND-Gattern ist ebenfalls möglich).



Wahrheitstabelle eines NOR-Gatters:

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0

Bild 4.2: SR-Flip-Flop

Das SR-FF hat einen **Setz**- und einen **Rücksetzeingang** (daher der Name). Seine Funktionsweise ergibt sich aus der Wertetabelle des hier verwendeten NOR-Gatters. Liegt am s-Eingang 1 und am r-Eingang 0, so wird $p=0$ (NOR-Gatter G1). NOR-Gatter G2 erhält dann an beiden Eingängen 0 und es wird $q = \bar{p} = 1$.

Entsprechend führt $s=0, r=1$ zum inversen Zustand $p = \bar{q} = 1$. Mit diesen beiden Eingangskombinationen kann das FF offenbar - unabhängig von seinem Zustand - gesetzt ($q=1$) oder rückgesetzt ($q=0$) werden.

Eine Eingangsbelegung $s=r=0$ führt zu Ausgangssignalen, die abhängig sind vom Zustand: der gegebene Zustand $p = \bar{q}$ bleibt erhalten. Demgegenüber erzwingt $s=r=1$ in jedem Falle $p=q=0$, dies ist der einzige Zustand, in dem $p \neq \bar{q}$ gilt! Aus diesen Betrachtungen folgt die *Funktionstabelle des SR-FF* (Tabelle 4.1) mit dem vorliegenden Zustand q^n, p^n und dem Folgezustand q^{n+1}, p^{n+1} aufgrund der angelegten Eingangsbelegung.

Eingangsvariablen		Zustandsvariablen		Ausgangsvariablen		Funktion als 1 Bit - Speicher
s	r	q^n	p^n	q^{n+1}	p^{n+1}	q^{n+1}
0	0	0	1	0	1	q^n Speichern des
0	0	1	0	1	0	q^n geg. Zustandes
0	1	0	1	0	1	0 Rücksetzen
0	1	1	0	0	1	0 Rücksetzen
1	0	0	1	1	0	1 Setzen
1	0	1	0	1	0	1 Setzen
1	1	0	1	0	0	- unzulässig
1	1	1	0	0	0	- unzulässig

Tabelle 4.1: Funktionstabelle des SR-FF

Der Übergang von der Eingangsbelegung $s=r=1$ auf $s=r=0$ ist allerdings unbestimmt. Es entsteht ein Zustand, der durch den Übergang der Eingangssignale bestimmt ist. Die folgende *Skizze der Binärsignale* verdeutlicht die FF-Funktion:

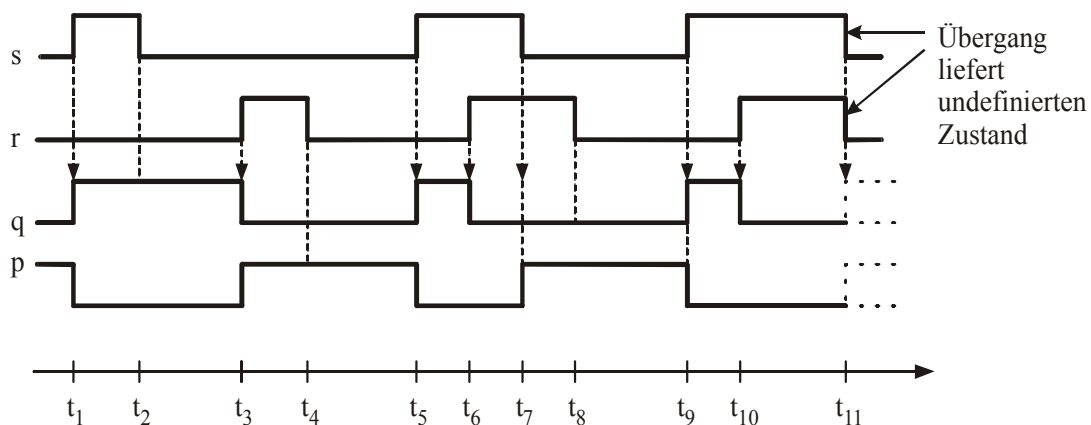


Bild 4.3: SR-Flip-Flop, Skizze der Binärsignale

Ersichtlich wird der entstehende Zustand aufgrund des Überganges von $s=r=1$ (bei t_{11}) auf $s=r=0$ abhängig sein von den Signalfanken der Eingangssignale bzw. von den Laufzeitunterschieden in den beiden Gattern aufgrund der Bauelementtoleranzen.

Aus der Funktionstabelle des SR-FF wird auch die Eigenschaft einer *1-Bit-Binärsignalspeicherung* deutlich: Mit $s=1$ oder $r=1$ wird ein einzelnes Bit eingeschrieben, da ja 2 definierte Speicherzustände möglich sind. Mit $s=r=0$ bleibt der Zustand erhalten. Eingangssignale $s=r=1$ führen zu unbestimmten Verhalten des SR-FF und sind daher auszuschließen. Auslesen ist immer möglich, der Speicherinhalt steht am Ausgang q (bzw. $\bar{q}$) zur Verfügung. Hier sind im übrigen die Zustandsvariablen identisch mit den Ausgangsvariablen.

4.2.1.1 Zustandsgraph zur Charakterisierung von Schaltwerken

Allgemein gilt für einen *endlichen Graphen*: er hat eine endliche Anzahl von Knoten und verbindenden Linien, sogenannten Kanten. *Knoten* werden durch Kreise symbolisiert und *Kanten* als Verbindungslinien zwischen den Knoten. Ein *gerichteter Graph* entsteht, wenn den Kanten Wirkungsrichtungen durch Pfeile zugewiesen werden. Eine Folge von Kanten ist ein *Weg*. Ein *zusammenhängender Graph* entsteht, wenn jeder Knoten von jedem anderen aus über mindestens einen Weg erreichbar ist. Ein in sich geschlossener Weg ist eine *Masche*. Eine Masche, bestehend aus einer Kante, die wieder im Ausgangsknoten endet, ist eine *Schlinge*. (Für eine ausführliche Behandlung der Graphentheorie siehe Literatur.)

Ein Zustandsgraph dient als visuelle Beschreibungsform des Übergangsverhaltens von Schaltwerken. Folgende Eigenschaften und Verabredungen kennzeichnen den Zustandsgraphen:

- (1) Ein Zustandsgraph ist ein gerichteter Graph.
- (2) Einem Knoten ist eindeutig ein stabiler Zustand zugeordnet. Im Knoten wird der Zustand (Zustandsvektor) eingetragen.
- (3) Einer gerichteten Kante ist eindeutig ein möglicher Zustandsübergang zugeordnet.
- (4) Pfeile kennzeichnen die Richtung des Zustandsüberganges.
- (5) Die Kantenbeschriftung enthält den übergangsbewirkenden Eingangsvektor und durch Schrägstrich getrennt den Ausgangsvektor (der zum Zustand gehört, von dem man aktuell ausgeht).
- (6) Zu jedem Knoten (Zustand) führt mindestens eine Kante (hin oder weg), es können aber auch alternativ mehrere Kanten von einem Knoten ausgehen

(entsprechend den möglichen Kombinationen bei mehreren Eingangsvariablen). Für einen eindeutigen Ablauf müssen sich die verschiedenen Möglichkeiten gegenseitig ausschließen.

- (7) Alle Abläufe im Zustandsgraph sind sequentiell, es gibt keine parallelen Läufe.
- (8) Bei k Zustandsvariablen gibt es höchstens 2^k Knoten (Zustände).
- (9) Bei m Eingangsvariablen können höchstens 2^m Kanten (Übergänge) von jedem Knoten ausgehen.

Der Zustandsgraph beschreibt hier das Verhalten der Schaltwerkes. Knoten repräsentieren die inneren Zustände der Schaltwerks, die gerichteten Kanten (Pfeile) stellen die Übergänge zwischen Zustände dar.

4.2.1.2 Zustandsgraph, Übergangstabelle und Übergangsgleichung des SR-FF

Auf der Basis der obigen Verabredungen ergibt sich der *Zustandsgraph* des SR-Flip Flops (Bild 4.4):

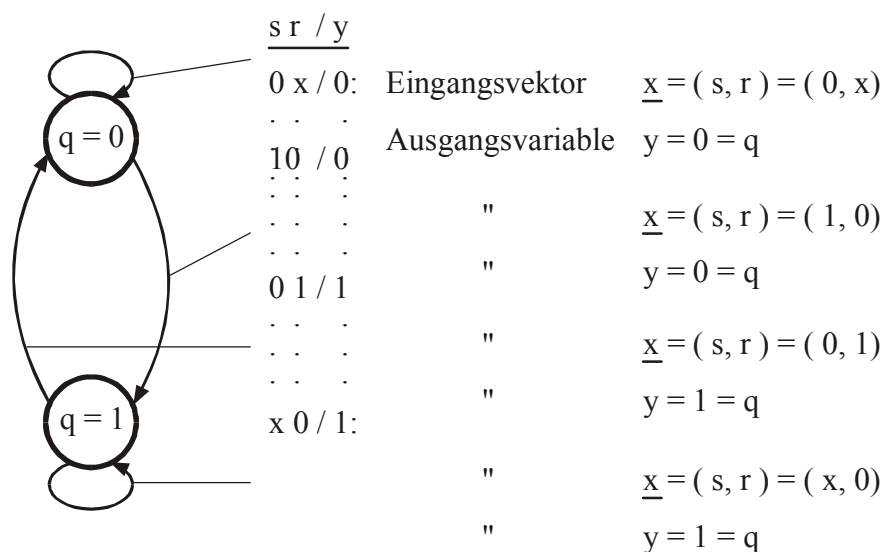


Bild 4.4: Zustandsgraph des SR-Flip Flops (In diesem Beispiel ist die Zustandsvariable q gleich der Ausgangsvariablen y .)

Der Zustandsgraph (auch Zustandsübergangsdiagramm) ist eine äquivalente Darstellung der *Übergangstabelle* (Funktionstabelle), die oben für das SR-FF bereits aufgestellt wurde. Aus dieser lässt sich mithilfe z. B. des KV-Diagrammes die *Übergangsgleichung* entwickeln (Bild 4.5):

s	r	q^n	q^{n+1}	$q^{n+1} :$
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	x	} verbotene Eingangsbedingungen, daher "don't care"
1	1	1	x	

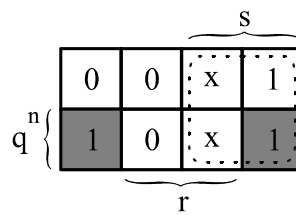


Bild 4.5: Übergangstabelle und KV-Diagramm

Hieraus folgt die Übergangsgleichung für das SR-FF:

$$q^{n+1} = s \vee \bar{r}q^n \quad (\text{mit } r=s=1 \text{ nicht zulässig})$$

Das Schaltsymbol (DIN 40900) des SR-FF zeigt Bild 4.6:

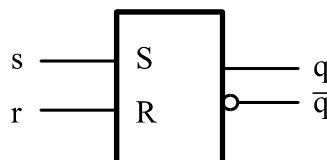


Bild 4.6: Schaltsymbol (DIN 40900) des SR-FF

4.2.2 Taktgesteuerte Flip Flops

Bislang wurden nur *asynchrone SR-Flip Flops* behandelt. Das sind Schaltungen, bei denen der Zustandswechsel unmittelbar nach Anliegen der Eingangssignalkombination erfolgt (transparente Flip Flops).

Häufig ist die Forderung jedoch gegeben, die Datensignale zu definierten Zeitpunkten aufgrund eines Taktsignales (Clockimpulses) zu übernehmen.

Dies führt zu taktgesteuerten Flip Flops mit *synchronem Schaltverhalten*.

Man unterscheidet

- taktzustandsgesteuerte Schaltungen
- taktflankengesteuerte Schaltungen

Im ersten Fall wird das an den Eingängen liegende Signal während der ganzen Dauer des Taktimpulses übernommen. Im zweiten Fall erfolgt die Übernahme nur während der auslösenden (aktiven) positiven oder negativen Taktflanke. Außerhalb dieser aktiven Taktflanken ist das FF unempfindlich gegenüber Signaländerungen und Störsignalen am Eingang.

Neben diesen einflankengesteuerten FF gibt es noch die wichtige Gruppe zweiflankengesteuerter FF. Hier wird das Eingangsdatensignal während der Taktimpulsdauer (bzw. z. B. der ansteigenden Flanke) übernommen, ist aber erst mit der aktiven (bzw. z. B. der abfallenden) Taktflanke am Ausgang verfügbar. Hierzu sind FF mit Zwischenspeicherung erforderlich.

4.2.2.1 Taktzustandsgesteuertes SR-Flipflop

Durch konjunktive Verknüpfung der beiden Eingänge des SR-FF mit je einem UND-Gatter erhält man ein *taktzustandsgesteuertes SR-FF* (Bild 4.7).

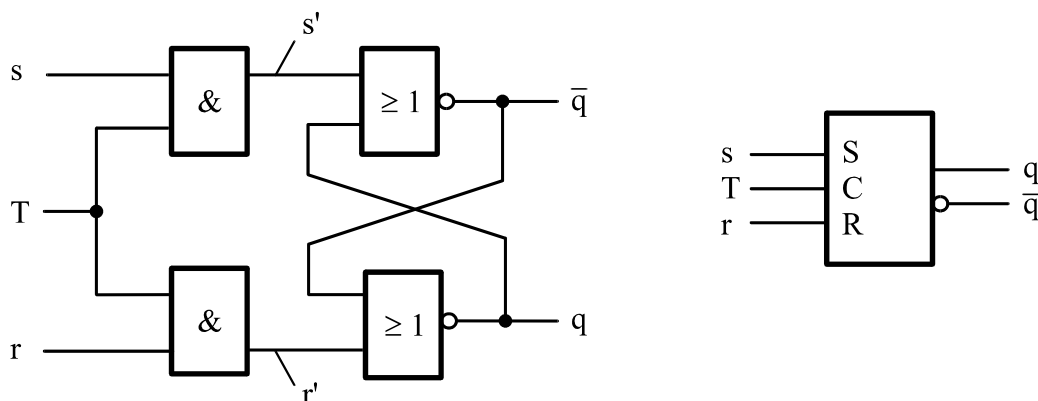


Bild 4.7: Taktzustandsgesteuertes SR-FF

Die Signale s , r werden übernommen, sobald der Takt T in den UND-Gattern die Übernahme freigibt. Die Eingänge werden zu "Vorbereitungseingängen". Außerhalb der Taktzeiten ist das FF gesperrt gegenüber Datensignaländerungen und Störungen. Das Signal-Zeit-Verhalten zeigt folgende Skizze:

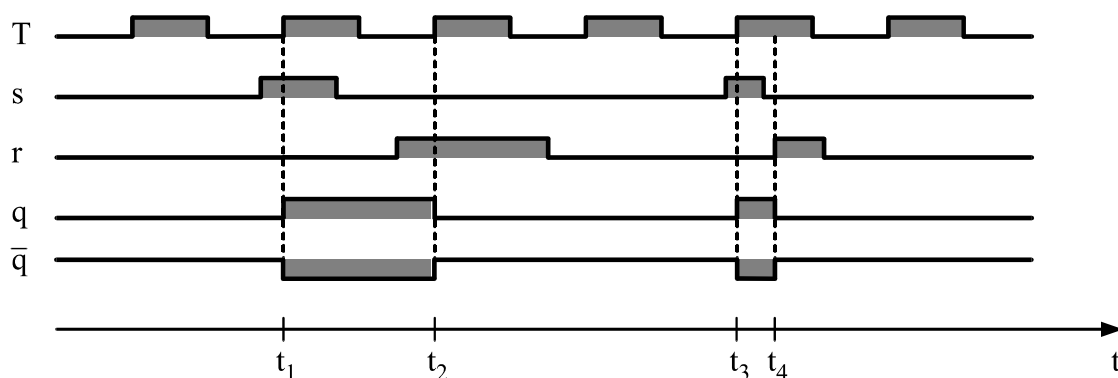


Bild 4.8: Signal-Zeit-Verhalten des taktzustandsgesteuerten SR-FF

Übergangstabelle und charakteristische Gleichung sind über die UND-Verknüpfung mit dem Takt leicht aus den Parametern des asynchronen SR-FF zu entwickeln

$$q^{n+1} = s' \vee \bar{r} q^n$$

$$= sT \vee \bar{r}Tq^n \quad \text{mit } s = r = 1 \text{ unzulässig}$$

Ebenso kann der Zustandsgraph leicht entsprechend modifiziert werden.

4.2.2.2 Taktzustandsgesteuertes D-Flipflop

Von Bedeutung ist in diesem Zusammenhang noch das *taktzustandsgesteuerte D-Flip Flop* (transparentes D-Flip Flop), s. Bild 4.9. Bei diesem wird $s = \bar{r}$ durch einen Inverter am r-Eingang erzwungen. Das Datensignal d wird dann mit dem Taktsignal übernommen. Eine Beschränkung bezüglich $s=r=1$ ist jetzt gegenstandslos.

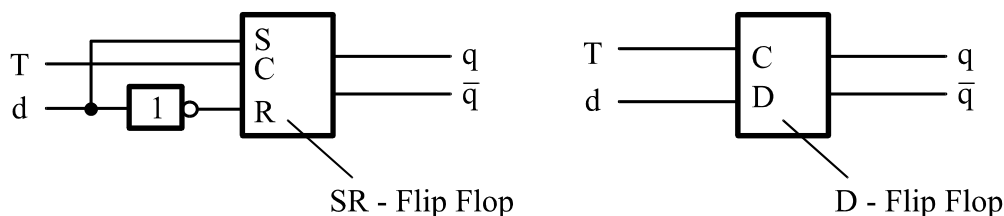


Bild 4.9: Taktzustandsgesteuertes D-Flip Flop

Die charakteristische Übergangsgleichung ist hier durch

$$q^{n+1} = d \wedge T$$

gegeben. Hieraus folgt leicht die Übergangstabelle.

4.2.2.3 Taktflankengesteuertes SR-Flipflop

Ein taktflankengesteuertes SR-Flip Flop erhält man z. B. durch das Einfügen eines Inverters in die Taktleitung (Bild 4.10) und geschicktes Ausnutzen der Gatterlaufzeit für eine Taktflankensteuerung.

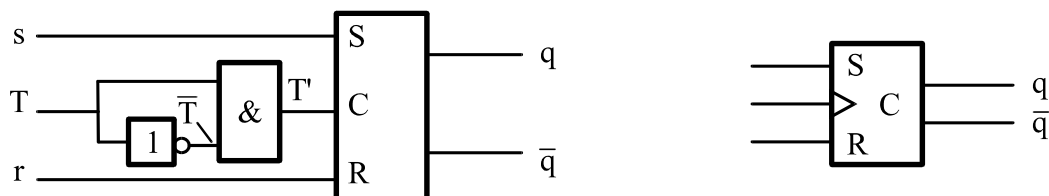


Bild 4.10: Taktflankengesteuertes SR-Flip Flop durch Einfügen eines Inverters

Die Funktion ist am folgenden Signal-Zeit-Diagramm (Bild 4.11) zu verdeutlichen:

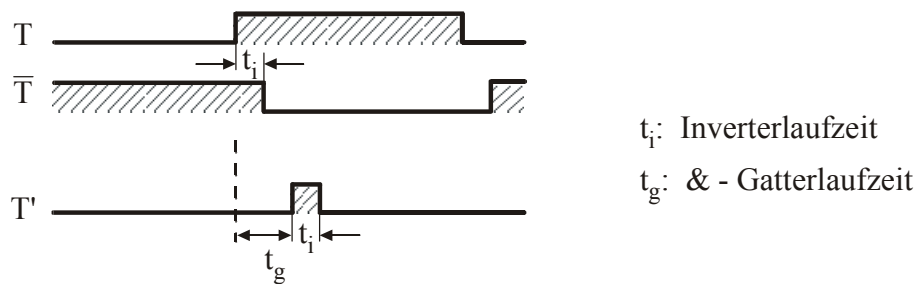


Bild 4.11: Signal-Zeit-Diagramm des taktflankengesteuerten SR-FF

Um die Inverterlaufzeit t_i verzögert ändert sich bei einer Taktflanke der komplementäre Takt $\bar{T}$. Um eine Gatterlaufzeit t_g verzögert erscheint daher ein durch die positive Flanke ausgelöster Taktimpuls T' am Ausgang des UND-Gatters. Nur während dieses Impulses ist das SR-Flip Flop zur Datenübernahme aktiviert.

4.2.2.4 Taktflankensteuerung durch Zwischenspeicher-Flipflops

In der integrierten Schaltungstechnik realisiert man gerne *taktflankengesteuerte FF* durch ein zusätzliches *Zwischenspeicher FF* (Bild 4.12). Das Prinzip sei am Beispiel eines SR-FF mit negativer Taktflankensteuerung - d. h. ein 1-0-Übergang des Taktes bewirkt eine Zustandsänderung - erläutert.

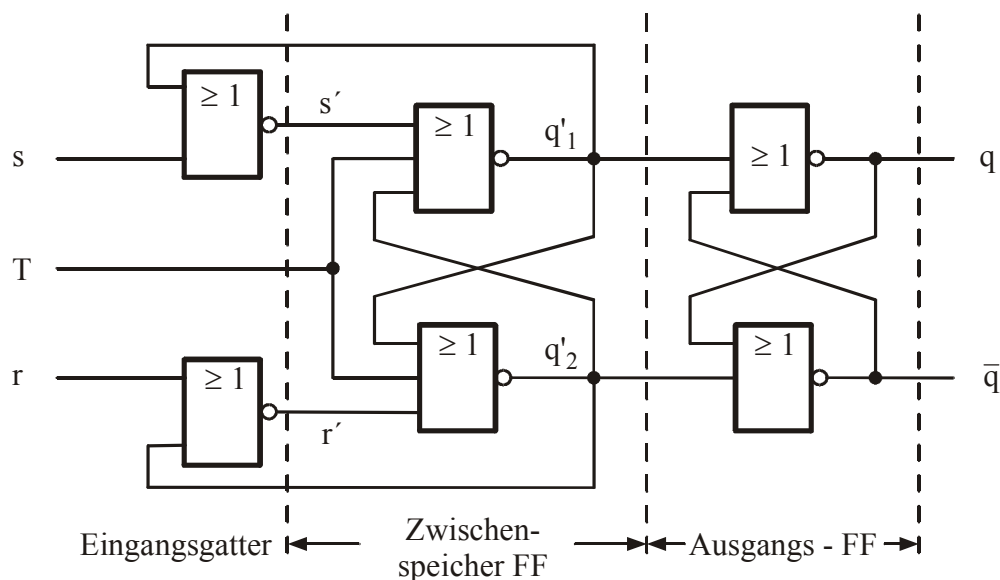


Bild 4.12: Taktflankengesteuerte FF durch Zwischenspeicher FF

Ist der Takt $T=1$ gegeben, so ist das Zwischenspeicher FF gesperrt, beide Ausgänge der Zwischenspeicher FF liegen auf 0. Das Ausgangs-FF hält somit seinen gegenwärtigen Zustand. Durch die Rückführung der Signale $q_1', q_2' = 0$ auf die Eingangsgatter sind diese jetzt durch s bzw. r schaltbar: s , bzw. $r=1$ erzeugt 0 am Ausgang des jeweiligen Eingangsgatters ($s'=0$, bzw. $r'=0$).

Die NOR-Gatter im Zwischenspeicher schalten genau dann auf 1, wenn eine Eingangskombination 000 vorliegt. Offensichtlich kann somit gesetzt bzw. rückgesetzt werden, wenn s bzw. $r=1$ gesetzt wird und gleichzeitig das Taktsignal sich von 1 auf 0 ändert. Der Zustand $q_1'=1=\bar{q}_2'$ bzw. $q_2'=1=\bar{q}_1'$ wird sofort vom Ausgangs-FF übernommen, andererseits auch sofort auf das Eingangsgatter übergeben. Dieses wird dadurch nach einer Gatterlaufzeit unabhängig von eventuellen Änderungen von s bzw. r .

Ersichtlich kann auf diese Weise ein (hier im Beispiel negativ) taktsflankengesteuertes FF realisiert werden, das erst nach Zurückgehen des Taktsignales T auf 1 erneut für eine Bitübernahme vorbereitet wird. Derartige Zwischenspeicher enthaltende FF sind sehr gut für den Aufbau von Zählern und Schieberegistern geeignet.

Taktsflankengesteuerte FF bezeichnet man auch als **dynamische FF**, *Schaltungssymbole* s. Bild 4.13.

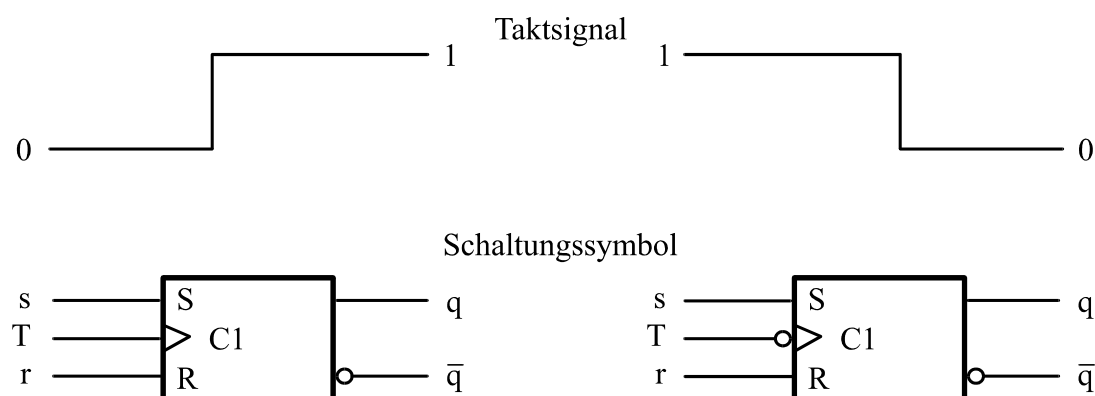


Bild 4.13: Dynamische FF, Schaltungssymbole

Für eine definierte Angabe der Übergangsgleichungen werden die logischen "Taktflanken"-Variablen T_p und T_n eingeführt mit

$$T_p = \begin{cases} 1 & \text{für } T = (0/1) \text{ (positive Flanke)} \\ 0 & \text{sonst} \end{cases}$$

$$T_n = \begin{cases} 1 & \text{für } T = (1/0) \text{ (negative Flanke)} \\ 0 & \text{sonst} \end{cases}$$

Für das **negativ flankengesteuerte SR-FF** wie beschrieben ergibt sich

$$q^{n+1} = sT_n \vee \overline{rT_n}q^n$$

mit $srT_n = 0$

Der Zustandsgraph ergibt sich entsprechend Bild 4.14 ($r=s=1$ verboten).

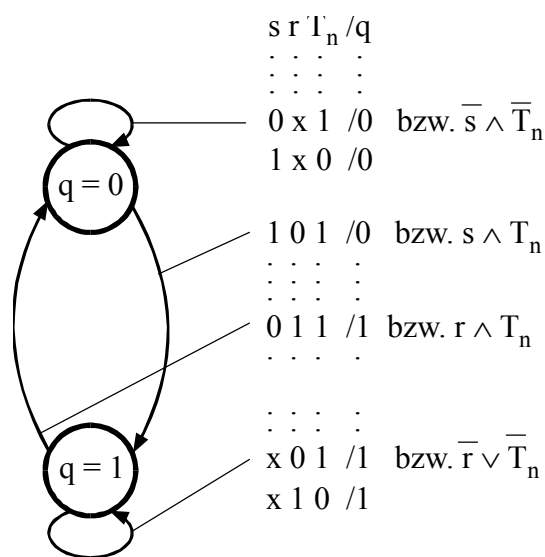


Bild 4.14: Zustandsgraph für das negativ taktflankengesteuerte SR-FF.

Bei synchronen Schaltwerken, bei denen eine Zustandsänderung nur zu den definierten Taktflanken möglich ist, empfiehlt es sich vereinfachend die Übergangsgleichung ohne Taktvariablen zu schreiben und die Änderungszeitpunkte im Zählindex n , $n+1$ zu berücksichtigen:

$$q^{n+1} = s \vee \overline{r}q^n \quad (s=r=1 \text{ unzulässig})$$

4.2.3 Master-Slave-Technik zur Zweizustandssteuerung

Wie bereits erläutert, besteht ein Schaltwerk aus einem kombinatorischen Schaltnetz und einem Rückkopplungspfad mit Speicherelementen. Voraussetzung für ein definiertes stabiles (Schnitt-) Verhalten des Schaltwerks ist allerdings, daß die rückgekoppelten Signale an den Ausgängen der Speicherelemente nicht wechseln solange noch deren Eingänge aktiv sind. Es werden daher Speicherelemente mit verzögerten Ausgangssignalen benötigt.

Bei den FF mit Zwischenspeichern wie beschrieben erfolgt bereits eine Trennung der Übernahme der Eingangsinformation und dem Schalten der FF-Ausgänge aufgrund der Gatterlaufzeiten. Allerdings dürfen sich auch hier die Eingangsdaten während der Einstell- und Übernahmezeit nicht ändern, da sonst mehrere Wechsel in einer FF-Kette möglich sind.

Übernahme und Weitergabe lassen sich günstig durch das Prinzip der Zweizustandssteuerung entkoppeln.

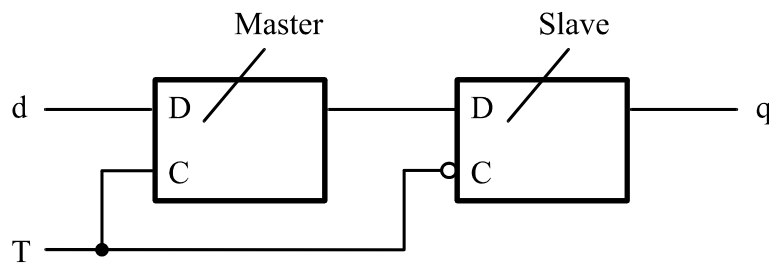


Bild 4.15: Master-Slave-FF

Dies ist möglich mithilfe von "Master-Slave"-FF. Ein Vorspeicher-FF (Master) übernimmt z. B. mit dem positiven Taktzustand die Eingangsvariable d und gibt sie mit der negativen Taktflanke an das Hauptspeicher-FF (Slave) weiter (Bild 4.15).

4.2.3.1 SR-Master-Slave-FF, 2-zustandsgesteuert

Ein SR-Master-Slave-FF kann aus zwei taktzustandsgesteuerten SR-FF aufgebaut werden (Bild 4.16).

Das Master SR-FF ist eingangsseitig positiv taktzustandsgesteuert. Während eines positiven Taktsignalzustandes kann es gesetzt bzw. rückgesetzt werden (vgl. Taktzeitdiagramm). Mit dem negativen Taktzustand ist das Master-FF verriegelt, so dass das Hauptspeicher-FF (Slave), das über einen invertierenden Takteingang negativ zustandsgesteuert ist, mit der negativen Taktflanke den Masterzustand übernimmt und hält. Das Schaltungssymbol eines solchen zweizustandsgesteuerten - in der Wirkung am Ausgang negativ flankengetriggerten - SR-MS-FF zeigt Bild 4.17.

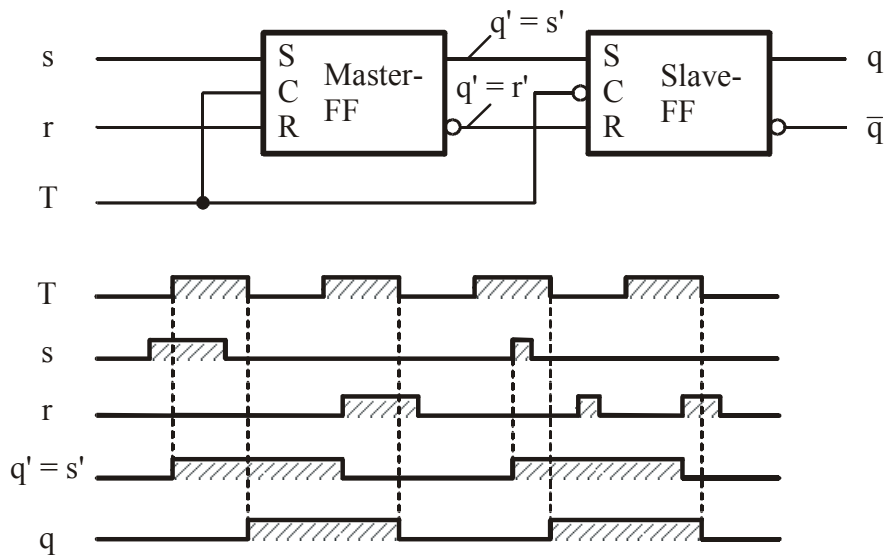


Bild 4.16: SR-Master-Slave-FF aus zwei taktzustandsgesteuerten SR-FF

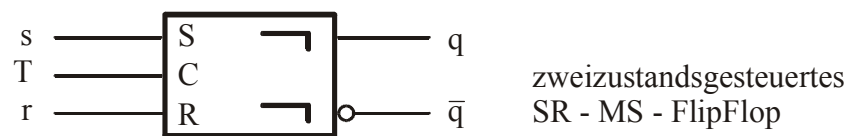


Bild 4.17: Zweizustandsgesteuertes SR-Master-Slave-FF

Als ein Anwendungsbeispiel sei ein *Schieberegister aus SR-Master Slave-FF* betrachtet (Bild 4.18).

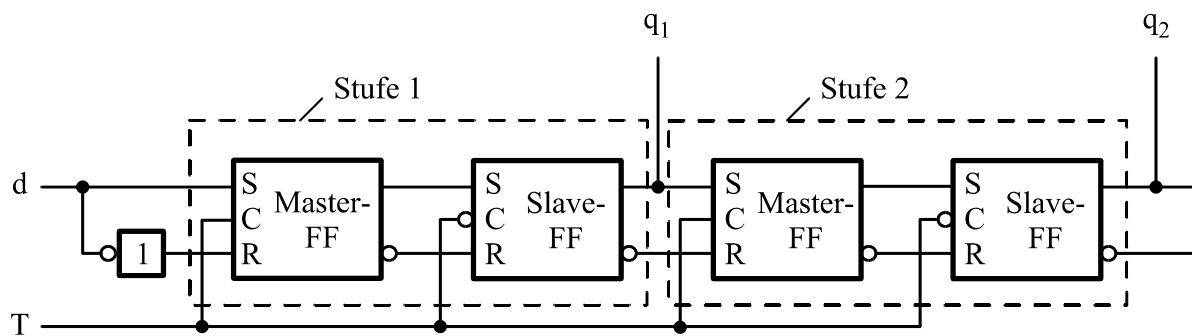


Bild 4.18: Schieberegister aus SR-Master Slave-FF

Mit positivem Taktzustand wird das Datensignal d in das Eingangs-FF übernommen und mit der negativen Taktflanke weitergeschoben. Die zweite Stufe übernimmt mit der positiven Taktflanke und stellt das Ausgangssignal mit der negativen Taktflanke

zur Verfügung etc. Auf diese Weise gelingt eine systematische Entkopplung und ein unabhängiges taktsynchrones Schalten der einzelnen Stufen.

4.2.3.2 JK-Master-Slave-FF, 2-zustandsgesteuert

Bei allen SR-Flip Flop-Typen ist die Eingangskombination $s=r=1$ verboten, da sie zu undefiniertem Verhalten führt. Im JK-Flip Flop (Bild 4.19) wird diese Einschränkung überwunden. Für diese Eingangsdaten wechselt das FF in seinen komplementären Ausgangszustand: $q^{n+1} = \overline{q^n}$.

Erreicht wird dies durch UND-Verknüpfung der Ausgangs- mit den Eingangssignalen.

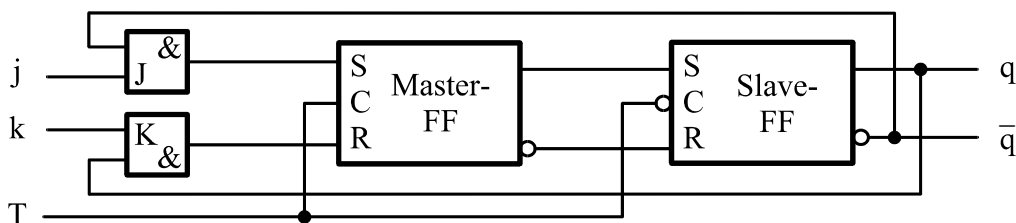


Bild 4.19: 2-zustandsgesteuertes JK-Master-Slave-FF

Ist beispielsweise $q = 0$, so ist während der 1-Phase des Taktes T der J-Eingang (wegen $\overline{q} = 1$) durchgeschaltet und es kann mit $j=1$ das Master-FF gesetzt werden. Ein Rücksetzen des Master-FF ist (im Gegensatz zum SR-MS-FF) nicht mehr möglich, da der K-Eingang (wegen $q = 0$) weiterhin gesperrt ist. Entsprechendes Verhalten ergibt sich für $q = 1$: dann kann wegen der Freigabe des K-Einganges rückgesetzt werden.

Mit der negativen Taktflanke wird der gesetzte (bzw. rückgesetzte) Zustand vom Slave-FF übernommen und es wird $q = 1$ (bzw. beim Rücksetzen: $\overline{q} = 1$). Offensichtlich ist stets einer der beiden Eingänge durch den Slave-Zustand geöffnet bzw. gesperrt, so dass der logische Zustand der Eingangsvariablen am gesperrten Eingang ohne Belang ist. $j=k=1$ kann daher zugelassen werden und bewirkt eine Negation des Ausgangszustandes. Aus diesen Betrachtungen folgen *Funktionstabelle*, *KV-Diagramm* (Bild 4.20) und *charakteristische Gleichung* für das JK-Flip Flop. Ebenfalls ist das *Schaltzeichen für das zwei-zustandsgesteuerte JK-FF (Master-Slave-Typ)* angegeben.

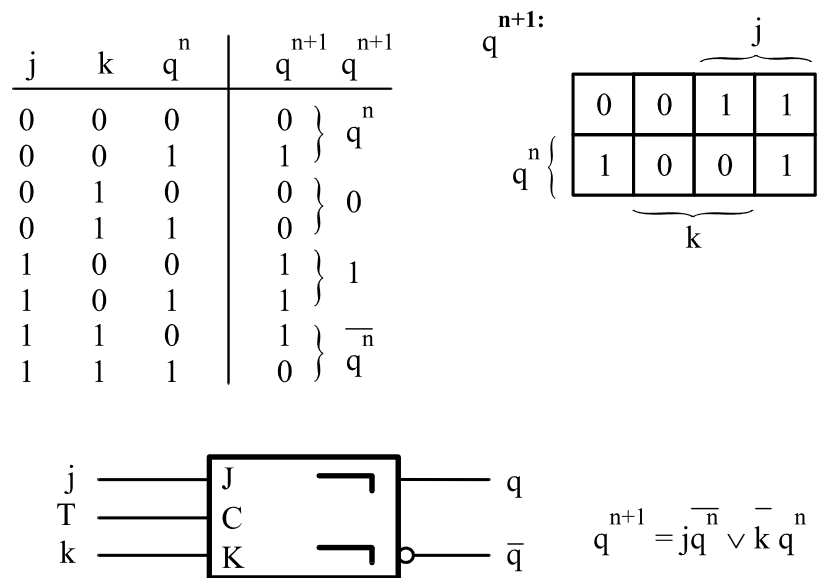


Bild 4.20: Funktionstabelle, KV-Diagramm und Schaltzeichen für das JK-FF

Der Zustandsgraph (Bild 4.21) des JK-Flip Flop ergibt sich aus folgenden Überlegungen

- | | | |
|-----------------------|-----------------------------------|------------------|
| (1) Zustandswechsel | $q^n = 0 \rightarrow q^{n+1} = 1$ | durch $j=1, k=x$ |
| (2) Zustandswechsel | $q^n = 1 \rightarrow q^{n+1} = 0$ | durch $j=x, k=1$ |
| (3) Zustandswechsel | $q^{n+1} = \overline{q^n}$ | durch $j=1, k=1$ |
| (4) Zustandserhaltung | $q^{n+1} = q^n$ | durch $j=0, k=0$ |

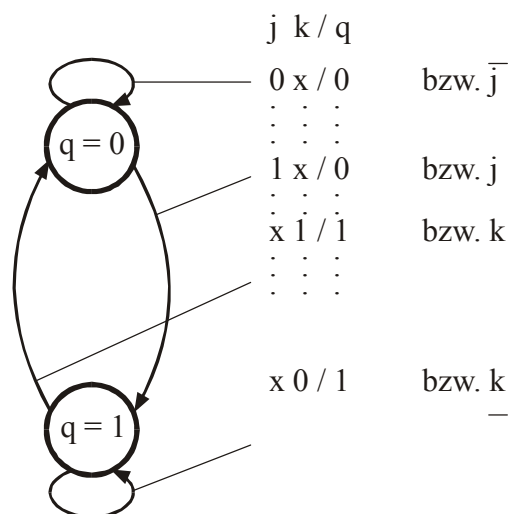


Bild 4.21: Zustandsgraph für das JK-FF

4.2.3.3 Master-Slave-FF mit Taktzustands- und Taktflankensteuerung

Die beschriebene 2-Taktzustandssteuerung soll hier nochmals am Beispiel eines SR-MS-FF genauer untersucht werden (Bild 4.22).

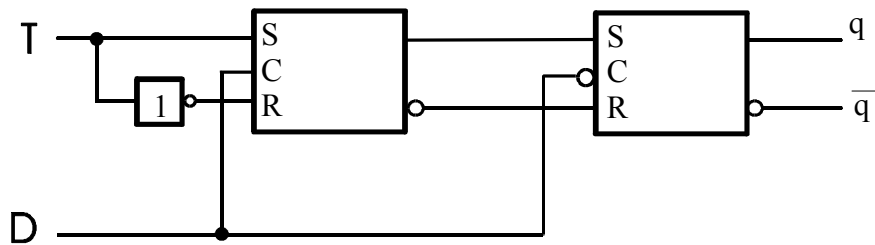


Bild 4.22: Master-Slave FF mit Taktzustandssteuerung

Das nachfolgende Diagramm (Bild 4.23) veranschaulicht die Vorgänge während eines Taktimpulses:

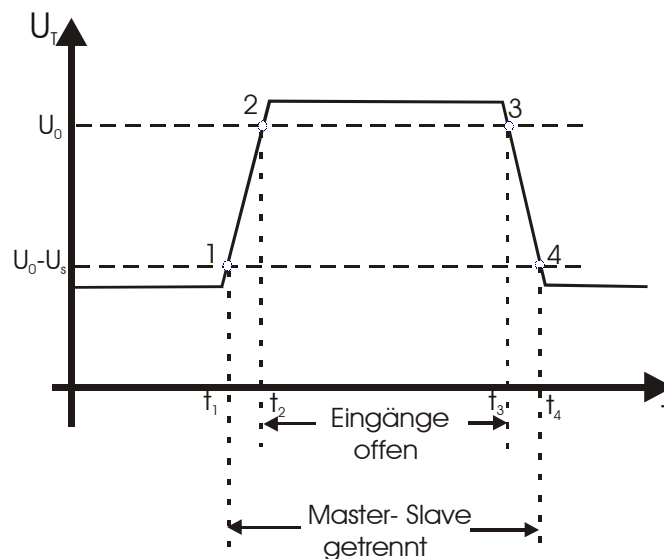


Bild 4.23: Triggersignal für Master Slave-FF

- t_1 : Master und Slave werden getrennt
- t_2 : Eingänge werden zum Master durchgeschaltet
- t_3 : Eingänge werden vom Master getrennt
- t_4 : Slave übernimmt Zustand vom Master

Man erkennt, daß für die Funktion eines Master-Slave-FFs vier Arbeitspunkte auf diesem Taktimpuls entscheiden sind. Es sind zwei Spannungswerte definiert, eine obere Schwelle U_0 und eine untere Schwelle $U_0 - U_s$. Unterhalb $U_0 - U_s$ (Punkt 1) sind

Master und Slave miteinander verbunden, oberhalb sind sie getrennt. An diesem Punkt beginnt bei Schaltung zu arbeiten.

Überschreitet das Taktsignal U_T die Schwelle U_0 (Punkt 2), dann wird die Information an den Eingängen S und R vom Master übernommen. Ein Signalwechsel an den Eingängen S, R kann während der gesamten Zeitdauer von Punkt 2 bis 3 in den Master übernommen werden. Bei Unterschreiten von U_0 (Punkt 3) werden die Eingänge R, S gesperrt und eine Änderung der Eingangssignale hat jetzt keine Auswirkung mehr auf den Zustand des MS-FFs. Hat das Taktsignal den Punkt 4 erreicht, so sind Master-FF und Slave-FF wieder verbunden. Die vom Master gespeicherte Information wird zum Slave weitergegeben und erscheint damit an den Ausgängen Q und $\bar{Q}$ des MS-FFs. Die Speicherung der Eingangssignale ist abgeschlossen. Neue Informationen können bei diesem SR-MS-FF wieder bei Ansteigen des Taktsignals in der oben beschriebenen Reihenfolge verarbeitet werden.

Bei Kippgliedern mit Taktzustandssteuerung, wie sie bisher beschrieben wurden, sind die Variablen am Eingang wirksam solange das Taktsignal $C=1$ ist. Damit ein solches Kippglied den geforderten Zustand einnimmt und beibehält, dürfen sich die Variablen am Eingang für die Plusdauer des Taktsignales $C=1$ nicht ändern und es dürfen in diesem Zeitraum keine Störsignale auftreten.

Eine Lösung dieses Problems ist möglich durch den Übergang zu **dynamischen MS-Flip Flops** mit einer Eingangsverriegelung (oben für einstufige FF) , so dass genau der Eingangszustand eingelesen wird, der (z. B.) bei der positiven Taktflanke anliegt. Diese Verarbeitungstechnik veranschaulicht das folgende Diagramm (Bild 4.24):

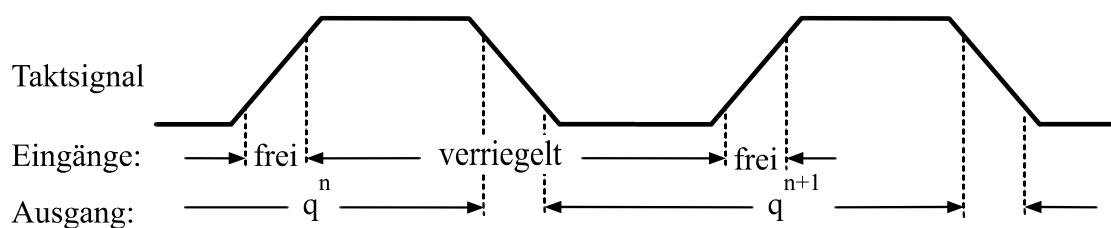


Bild 4.24: Taktsteuerung bei dynamischen MS-FF mit Eingangsverriegelung

Während der Taktzustände $C=1$ bzw. $C=0$ sind die Variablen am Eingang nicht wirksam, das gilt ebenso für Störsignale in diesem Zeitintervallen. Das Schaltsymbol z. B. eines derartigen positiv 2-flankengesteuerten JK-Flip Flops zeigt Bild 4.25:

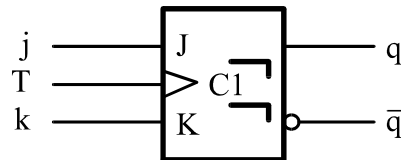


Bild 4.25: Schaltsymbol eines positiv 2-flankengesteuerten JK-Flip Flops

Betrachtet man synchrone Schaltwerke, die mit derartigen flankengesteuerten Speichern arbeiten, so werden in der Regel die logischen Taktverknüpfungen (charakterist. Gleichung, Tabelle etc.) weggelassen und die logischen Variablen nur zu den Taktzeitpunkten betrachtet.

4.2.3.4 Toggle-MS-Flip Flop

Durch Verbinden beider Eingänge des JK-Flip Flop erhält man ein "Toggle"-FF (Bild 4.26), das solange bei jeder aktiven Taktflanke kippt, wie am Eingang eine 1 anliegt.

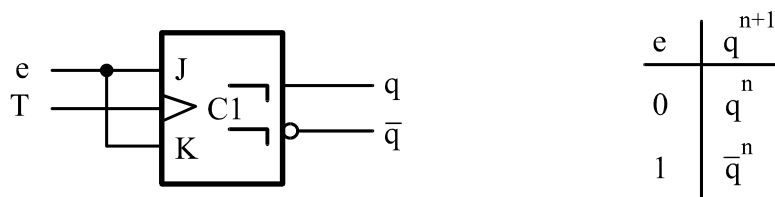


Bild 4.26: "Toggle"-FF

Die charakteristische Gleichung ist modifiziert, es ergibt sich

$$q^{n+1} = j\bar{q}^n \vee \bar{k}q^n \quad \text{mit } j = k = e$$

$$q^{n+1} = e\bar{q}^n \vee \bar{e}q^n$$

Das Signalzeitdiagramm (Bild 4.27) verdeutlicht das Frequenzteilverhalten (Modulo2-Zähler) bei positiver Flankensteuerung bei der Signalübernahme.

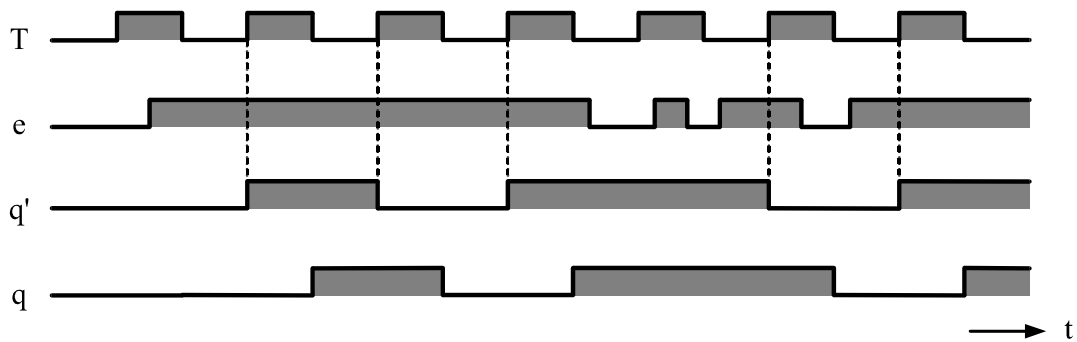


Bild 4.27: Signalzeitdiagramm beim Toggle-FF

4.2.3.5 D-Master-Slave-FF

Durch Zuführung des invertierten j-Signals auf den k-Eingang erhält man ein D-MS-FF (Bild 4.28) mit dem Eingangssignal

$$j = \bar{k} = d$$

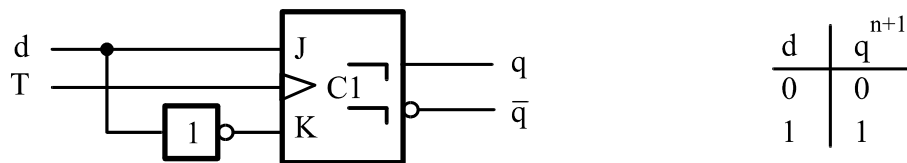


Bild 4.28: D-MS-FF

Es folgt die charakteristische Gleichung mit $j = \bar{k} = d$

$$q^{n+1} = j\bar{q}^n \vee \bar{k}q^n$$

$$q^{n+1} = d\bar{q}^n \vee dq^n = d$$

Das heißt: Mit jeder aktiven Taktflanke übernimmt das D-FF das am Eingang liegende Datensignal d, vgl. Signalzeitdiagramm (Bild 4.29) für positive Flankensteuerung.

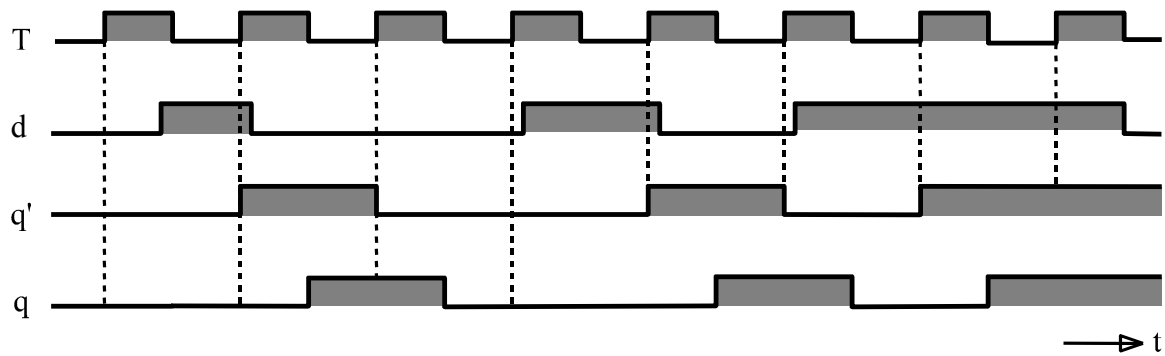


Bild 4.29: D-MS-FF, Signalzeitdiagramm für positive Flankensteuerung.

4.2.4 Arten zweiflankengesteuerter Flip Flops

In der Übersicht (Bild 4.30) sind positiv taktflankengesteuerte (0-1-Übergang) FF-Typen zusammengestellt mit einer Übernahme der Daten am Ausgang bei den Taktflanken (MS-Flip Flops).

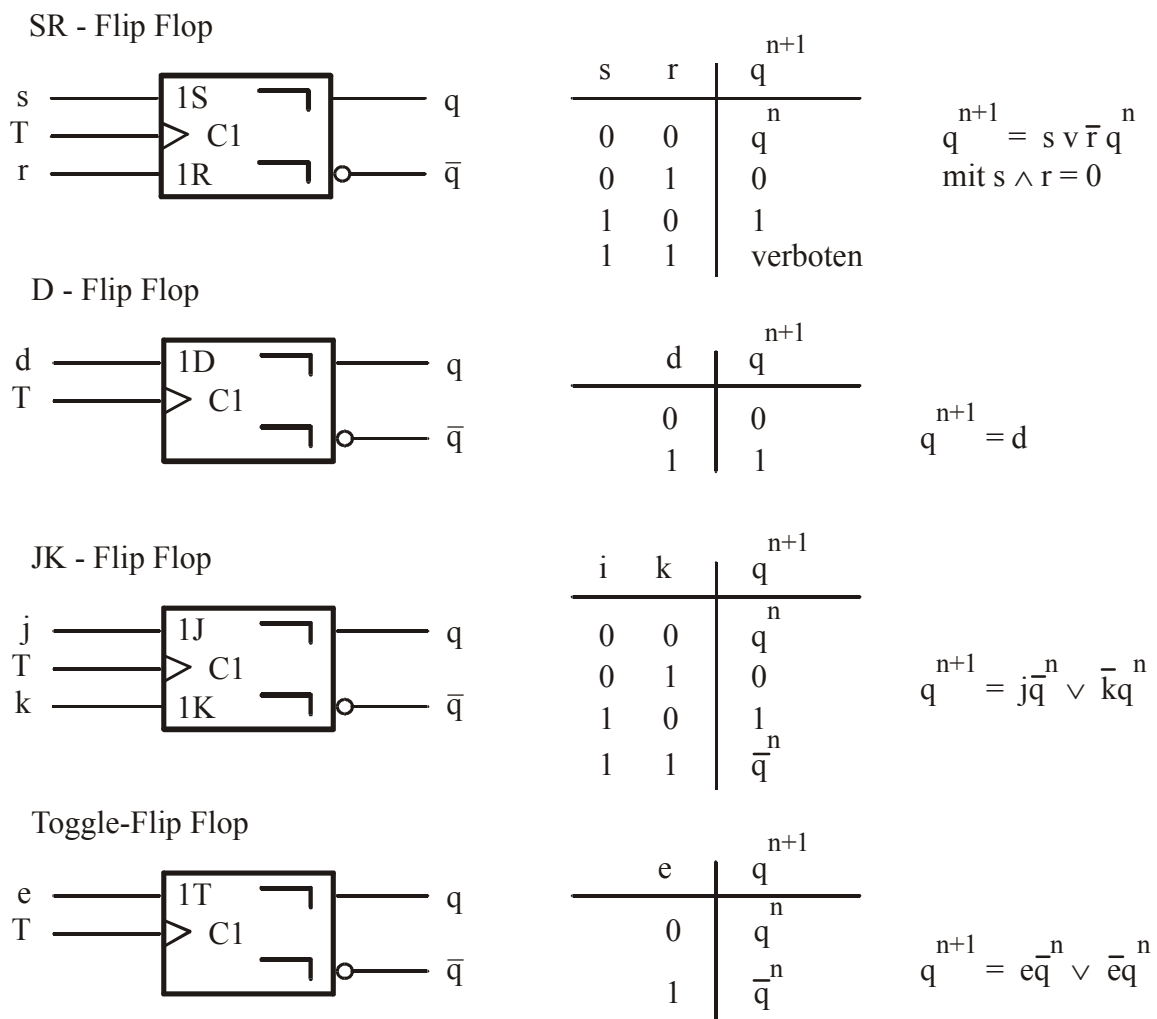


Bild 4.30: Arten zweiflankengesteuerter Flip-Flops

4.3 Formale Schaltwerkbeschreibung

Das allgemeine Blockschaltbild eines Schaltwerkes war eingeführt worden entsprechend folgender Skizze (Bild 4.31):

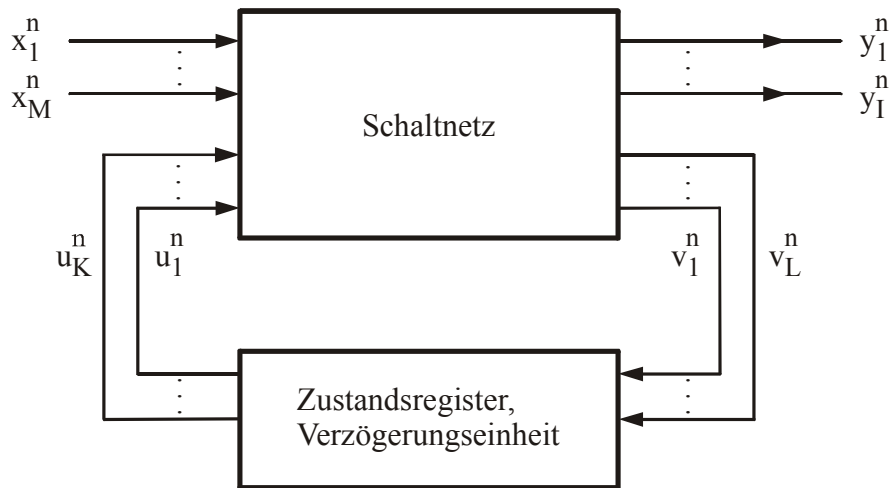


Bild 4.31: Schaltwerk, Blockschaltbild

Im Unterschied zu den rein kombinatorisch arbeitenden Schaltnetzen sind die Ausgänge des Schaltwerkes von früheren Eingangssignalen bzw. Zuständen des Schaltwerkes abhängig. Es handelt sich also um eine sequentiell verschiedene Zustände durchlaufende logische Schaltung. Zur Kennzeichnung dieser zeitlichen Abhängigkeit wurde der hochgestellte Sequenzindex n eingeführt. Es gilt für den

- Eingangsvektor: $\underline{x}^n = \{x_1^n, x_2^n, \dots, x_M^n\}$
- Ausgangsvektor: $\underline{y}^n = \{y_1^n, y_2^n, \dots, y_I^n\} = f(\underline{x}^n, \underline{x}^{n-1}, \dots, \underline{x}^{n-r})$
- Zustandsvektor: $\underline{u}^n = \{u_1^n, u_2^n, \dots, u_K^n\}$
- Registerkontrollvektor: $\underline{v}^n = \{v_1^n, v_2^n, \dots, v_L^n\}$

Der Zustand eines Schaltwerkes ist gekennzeichnet durch die Werte der intern gespeicherten Variablen, die Zustandsvariablen bzw. den Zustandsvektor $\underline{u}^n$ zum Zeitpunkt n . Der Folgezustand ist dann sinngemäß durch $\underline{u}^{n+1}$ beschrieben. Eingangsvektor und Ausgangsvektor bestimmen nun einerseits den Ausgangsvektor, andererseits den Folgezustand. Die Verknüpfungen sind gegeben durch Boolesche Funktionen:

- Ausgangsfunktion: $\underline{y}^n = f(\underline{x}^n, \underline{u}^n)$
- Registerkontrollfunktion: $\underline{v}^n = g_2(\underline{x}^n, \underline{u}^n)$
- Übergangsfunktion: $\underline{u}^{n+1} = g_1(\underline{v}^n)$

Registerkontrollfunktion und Ausgangsfunktion sind von den Eingangs- und Zustandsvariablen zum Zeitpunkt n abhängig. Die Übergangsfunktion ist allein eine Funktion der Registerkontrollvariablen. Sie bestimmt den Folgezustand, der zusammenfassend auch durch

$$\text{Übergangsfunktion: } \underline{u}^{n+1} = g_1(g_2(\underline{x}^n, \underline{u}^n)) = g(\underline{x}^n, \underline{u}^n)$$

dargestellt werden kann. Die Ausführungsform der Verzögerungseinheit bestimmt den Charakter des Schaltwerkes. Folgende Varianten können u. a. unterschieden werden:

- (1) Laufzeitglieder für die einzelnen Registerkontrollkomponenten jeweils
- (2) asynchron getaktete Flip Flop-Register, flanken- oder zustandsgesteuert z. B. SR-, JK- oder D-Flip Flop
- (3) synchron getaktete Flip Flop-Register, flanken- oder zustandsgesteuert z. B. SR-, JK- oder D-Flip Flop

Beispiele für die angegebenen Fälle zeigt folgende Skizze (Bild 4.32)

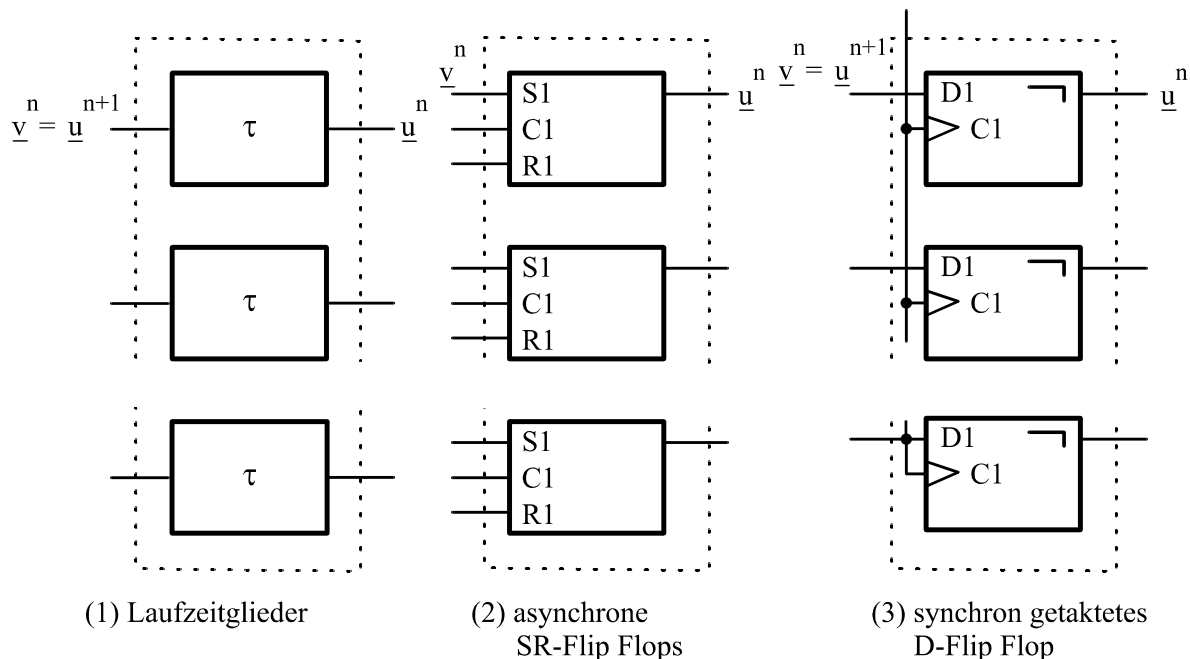


Bild 4.32: Ausführungsformen für Verzögerungselemente

Ein grundsätzliches Klassifizierungsmerkmal wird aus diesen Beispielen deutlich: man unterscheidet **asynchrone und synchrone Schaltwerke**. Bei einem asynchronen Schaltwerk ändern sich die Ausgangs- und Zustandsvariablen durch unmittelbare Reaktion auf Änderungen der Eingangsvariablen - nicht aber, wie bei synchronen Schaltwerken, aufgrund eines externen Taktsignals.

Weitere Vereinfachungen bei Schaltwerken sind häufig gegeben und sollen an dieser Stelle zusammengefasst werden:

(A) Registerkontrollvektor ist gleich dem Zustandsvektor zum Zeitpunkt (n+1):

$$\underline{v}^n = \underline{u}^{n+1}$$

In diesem Fall vereinfachen sich die Beschreibungsgleichungen:

- Ausgangsfunktion: $\underline{y}^n = f(\underline{x}^n, \underline{u}^n)$
- Übergangsfunktion: $\underline{u}^{n+1} = g(\underline{x}^n, \underline{u}^n)$

Ein Registerkontrollvektor ist dann überflüssig (z.B. bei Verwendung von D-FF oder Verzögerung um τ).

(B) Zusätzlich ist der Zustandsvektor zum Zeitpunkt (n+1) gleich dem Ausgangsvektor: $\underline{u}^{n+1} = \underline{y}^n$ (z.B. bei Zähler).

Es ergibt sich:

- Ausgangsfunktion: $\underline{y}^n = f(\underline{x}^n, \underline{u}^n) = \underline{u}^{n+1}$

Die Übergangsfunktion ist gleich der Ausgangsfunktion

(C) Das Schaltwerk ist von außen nicht beeinflussbar: **autonomes Schaltwerk**.

Es ergibt sich (zusätzlich ist Eigenschaft (A) angenommen):

- Ausgangsfunktion: $\underline{y}^n = f(\underline{u}^n)$
- Übergangsfunktion: $\underline{u}^{n+1} = g(\underline{u}^n)$

4.3.1 Asynchrone Schaltwerke

Formal können synchrone und asynchrone Schaltwerke mit den dargestellten Grundgleichungen in gleicher Weise behandelt werden. Bei asynchronen Schaltwerken sind jedoch die ggf. vorhandenen Taktsteuersignale logisch mit zu berücksichtigen, sie sind Teil der verschiedenen Vektoren. Hieraus ergibt sich, daß asynchrone Schaltwerke in der **Übergangsphase** von einem (stabilen) Zustand zum nächsten mehrere unterschiedliche Übergangszustände aufgrund von Verzögerungszeiten durchlaufen können. In dieser Zeit darf der Eingangsvektor sich nicht ändern. Mit dem Übergang irgendeiner Variable von z^n auf z^{n+1} ist genau eine Änderung nach Ablauf der Übergangsphase beschrieben.

Als Beispiel kann ein (*ungetaktetes*) *D-Flip Flop* als *asynchrones Schaltwerk* beschrieben werden. Dies sei in elementarer Form mit rückgekoppelten NOR-Gattern aufgebaut (Bild 4.33)

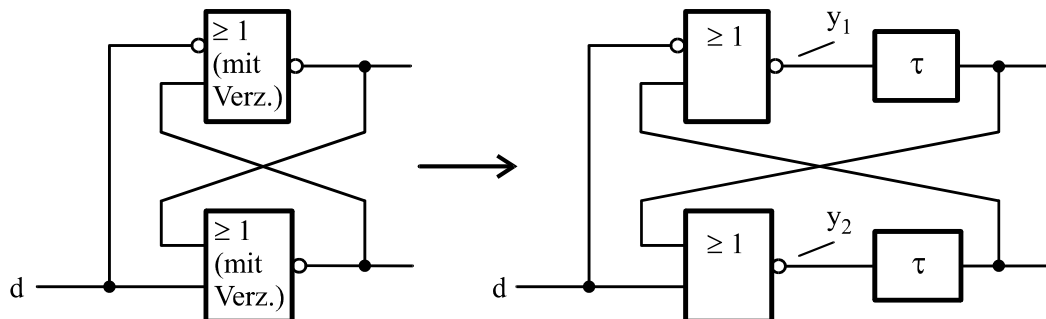


Bild 4.33: (Ungetaktetes) D-Flip Flop als asynchrones Schaltwerk

Die beiden Gatter haben jeweils konzentriert angenommene Verzögerungen. Es resultiert das folgende Ersatzbild (Bild 4.34).

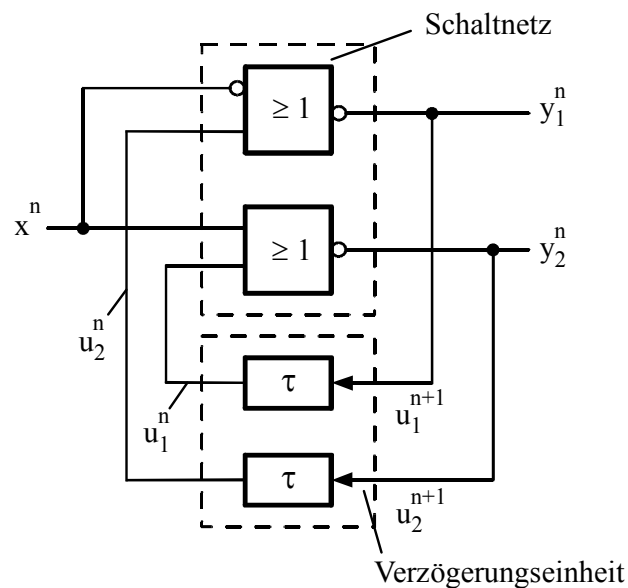


Bild 4.34: Ersatzbild für D-FF aus NOR-Gattern

Die *Zustands-(Übergangs-)tabelle* kann unmittelbar aus dieser Schaltung entwickelt werden (Bild 4.35). Es ergibt sich offenbar je ein stabiler Zustand für die Kombinationen

$$\{y_1^n, y_2^n\} = \{u_1^n, u_2^n\} = \begin{cases} \{0,1\} & \text{für } x^n = 0 \\ \{0,1\} & \text{für } x^n = 1 \end{cases}$$

x^n	u_1^n	u_2^n	Übergänge	u_1^{n+1}	u_2^{n+1}
				$= y_1^n$	$= y_2^n$
0	0	0		0	1
0	0	1		0	0
0	1	0		0	1
0	1	1		0	0
1	0	0		1	0
1	0	1		0	0
1	1	0		1	0
1	1	1		0	0

Bild 4.35: Zustands-(Übergangs-)tabelle für D-FF aus NOR-Gattern

Auf dem Wege in einen stabilen Zustand werden dabei verschiedene Übergangszustände (Races) durchlaufen (Übergangsphase bzw. Einschwingzeit). Die beschreibenden Gleichungen ergeben sich wie folgt:

Ausgangsgleichung :

$$y_1^n = x^n \overline{u_1^n} u_2^n \vee x^n u_1^n \overline{u_2^n} = x^n \overline{u_2^n}$$

$$= \overline{x^n \vee u_2^n} = u_1^{n+1}$$

Übergangsgleichung :

$$y_2^n = \overline{x^n} \overline{u_1^n} u_2^n \vee x^n \overline{u_1^n} \overline{u_2^n} = \overline{x^n} \overline{u_1^n}$$

$$= \overline{x^n \vee u_1^n} = u_2^{n+1}$$

Dadurch sind auch die instabilen Übergänge mitbeschrieben. Beschränkt man sich auf die stabilen Zustände, so reduziert sich die Beschreibung in die im Prinzip bekannte Gleichung

$$y_1^n = u_1^{n+1} = x^n$$

Asynchrone Schaltwerke diese Typs sind insgesamt betrachtet inzwischen von geringerer Bedeutung als synchrone Schaltwerke, weil hochkomplexe Schaltwerke mit ihren dann unübersichtlichen Übergangsphasen (Races!) schwierig zu entwickeln und zu realisieren sind. Aus den Übergangsphasen ergeben sich im übrigen die Schaltzeiten des Schaltwerkes und damit die maximale Betriebsfrequenz. Asynchrone Schaltwerke können daher dazu dienen, Modelle zu finden, die eine Beschreibung solcher Übergangsphasen z.B. in Hochgeschwindigkeitsanwendungen gestatten. Im übrigen werden die erwähnten Probleme durch synchrone Schaltwerke vermieden.

4.3.2 Synchrone Schaltwerke

Kennzeichen synchroner Schaltwerke ist ein *externer* dem Schaltwerk *aufgeprägter* (nicht von Zuständen oder Eingangs-, Ausgangsvektoren abhängiger) *Takt*, zu dessen Taktflanke (positiv oder negativ) eine Änderung der Zustandsspeicher zugelassen wird. Eine Änderung des Zustandsvektors $\underline{u}$ aufgrund sich asynchron ändernder Eingangsvariablen ist dann nur zu den durch die Taktflanken definierten Zeitpunkten möglich.

Ein solches synchrones Schaltwerk hat keine Übergangsphasen wie das asynchrone Schaltwerk, sondern ausschließlich stabile Zustände. Zwar ändern sich - wie bei asynchronen Schaltwerken - Ausgangsvektor und Registerkontrollvektor aufgrund von Eingangsvektoränderungen unmittelbar, jedoch folgt der Zustandsvektor diesen Änderungen erst mit der nächsten aktiven Taktflanke. Wegen dieser Eigenschaft, daß instabile Zwischenzustände nicht durchlaufen werden können, ist der Entwurf und die Realisierung synchroner Schaltwerke sehr viel einfacher. Daher haben synchron getaktete Schaltwerke insbesondere für getaktete Steuerwerke aller Art - z. B. als speicherprogrammierte Steuerungen bzw. in Verbindung mit Rechnern als Leitwerke - eine große Bedeutung gefunden.

4.3.2.1 Frequenzteiler (Modulo-4-Zähler)

Als erstes Beispiel soll die Funktion des synchronen Schaltwerkes an einem einfachen Frequenzteiler (Modulo 4-Zähler) deutlich gemacht werden (s. Bild 4.36). Verwendung finden hier 2 JK-Flip Flops.

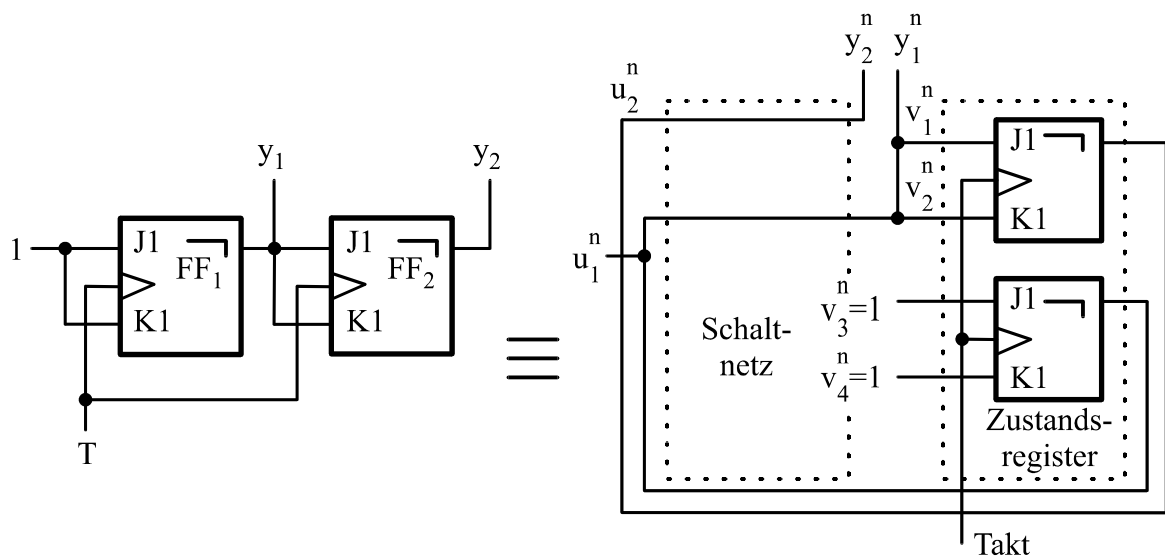


Bild 4.36: Beispiel: Frequenzteiler (Modulo 4-Zähler)

Es handelt sich um ein autonomes Schaltwerk, das nicht durch Eingangsvektoren beeinflusst werden kann.

Die Zustandstabelle kann unmittelbar angegeben werden, s. Bild 4.37. Das Signalzeitdiagramm zeigt das typische Frequenzteilverhalten.

u_2^n	u_1^n	v_2^n	v_1^n	$v_3^n = v_4^n$	u_2^{n+1}	u_1^{n+1}	y_2^n	y_1^n
0	0	0	0	1	0	1	0	0
0	1	1	1	1	1	0	0	1
1	0	0	0	1	1	1	1	0
1	1	1	1	1	0	0	1	1

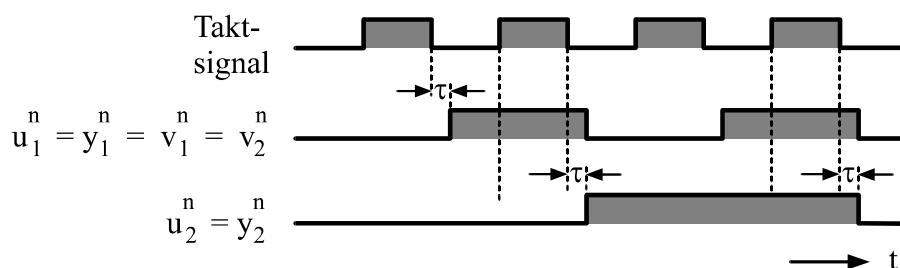


Bild 4.37: Zustandstabelle und Signal-Zeit-Diagramm für den Frequenzteiler

Mit der positiven Taktflanke wird das Registerkontrollsignal $v_1^n = v_2^n$ registriert und mit der negativen Taktflanke der Zustandsvektor geändert (Master-Slave-Prinzip). Für

beide FF ist eine gleiche Verzögerung um τ angenommen. Es gelten die Beschreibungsgleichungen:

Ausgangsfunktion $\underline{y}^n = \underline{u}^n$ (Ausgangsvektor = Zustandsvektor)

Registerkontrollfunktion $v_1^n = v_2^n = u_1^n$, $v_3^n = v_4^n = 1$

Übergangsfunktion

$$u_1^{n+1} = \overline{u_1^n}$$

$$u_2^{n+1} = v_1^n \overline{u_2^n} \vee \overline{v_1^n} u_2^n$$

$$= u_1^n \overline{u_2^n} \vee \overline{u_1^n} u_2^n$$

(zum Vergleich das JK-Flip Flop: $q^{n+1} = \overline{j}q^n \vee \overline{k}q^n$)

Der zugehörige Zustandsgraph hat 4 stabile Zustände, die von außen unbeeinflussbar, d.h. autonom, durchlaufen werden. Die Zustände sind mit $\underline{u}^n = \{u_2^n, u_1^n\}$ im Graphen (Bild 4.38) spezifiziert.

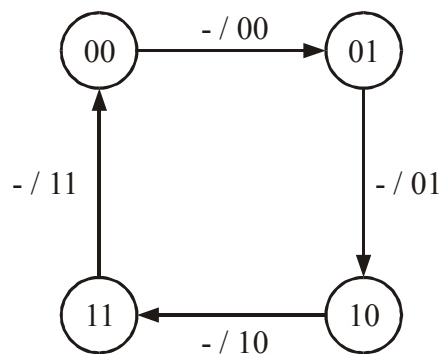


Bild 4.38: Übergangsgraph für den Frequenzteiler

4.3.2.2 Extern einstellbares Modulo-2-Zählwerk

Als weiteres Beispiel für synchrone Steuerwerke sei ein extern einstellbares Modulo 2-Zählwerk betrachtet. Das Zählwerk hat $i=2$ Zustände, es ist mit einem ($m=1$) Zustandsspeicher (allgemein: $m = \text{Id } i$) in der Rückführung realisierbar. Von außen seien folgende Funktionen einstellbar:

- Stillstand mit Reset (Nullsetzen des Zustandsspeichers)

- Stillstand mit Speicherung des aktuellen Zustands
- Stillstand mit Voreinstellung (Einsetzen des Zustandsspeichers)
- Zählen

Der Einfachheit halber seien die Ausgangssignale gleich dem aktuellen und dem folgenden Zustandssignal: $\underline{y}^n = \{u^{n+1}, u^n\}$.

Vier verschiedene Eingangsfunktionen erfordern 2 Eingangsvariable x_1^n, x_2^n . Hiermit können formal Zustandstabelle und Zustandsgraph entwickelt werden (Bild 4.39).

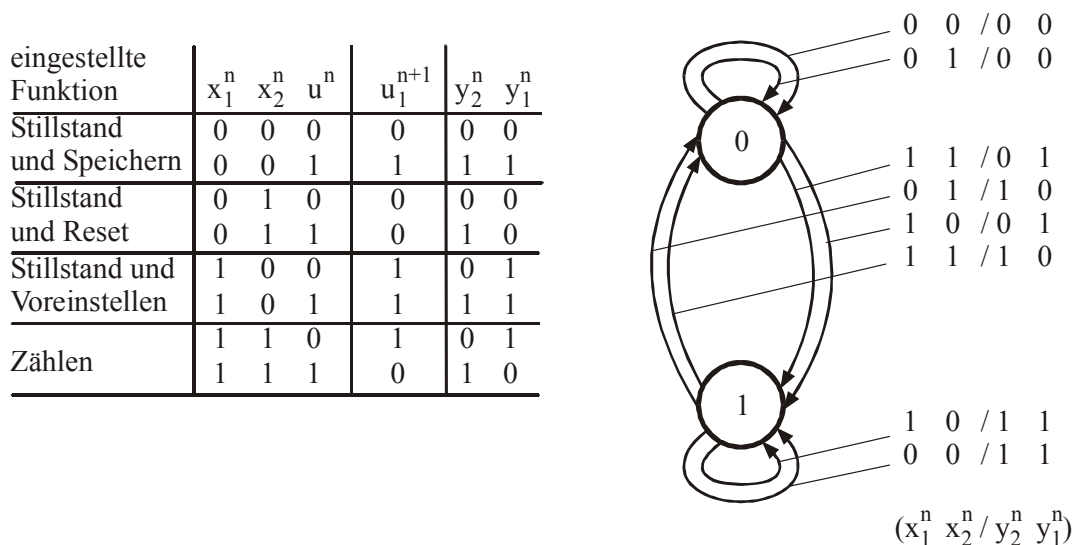
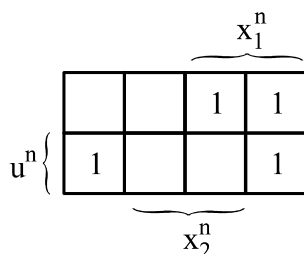


Bild 4.39: Zustandstabelle und -graph für extern einstellbares Modulo2-Zählwerk

Über ein einfaches KV-Diagramm erhält man durch Minimierung Ausgangsfunktion und Übergangsfunktion (Bild 4.40).

u^{n+1} :



$$u^{n+1} = x_1^n \overline{u^n} \vee \overline{x_2^n} u^n$$

$$y_1^n = u^{n+1} = x_1^n \overline{u^n} \vee \overline{x_2^n} u^n$$

$$y_2^n = u^n$$

Bild 4.40: KV-Diagramm für extern einstellbares Modulo2-Zählwerk

Mit einem *D-Flip Flop* als Zustandsspeicher kann eine Schaltungsrealisierung aus den Gleichungen direkt abgelesen werden (Bild 4.41).

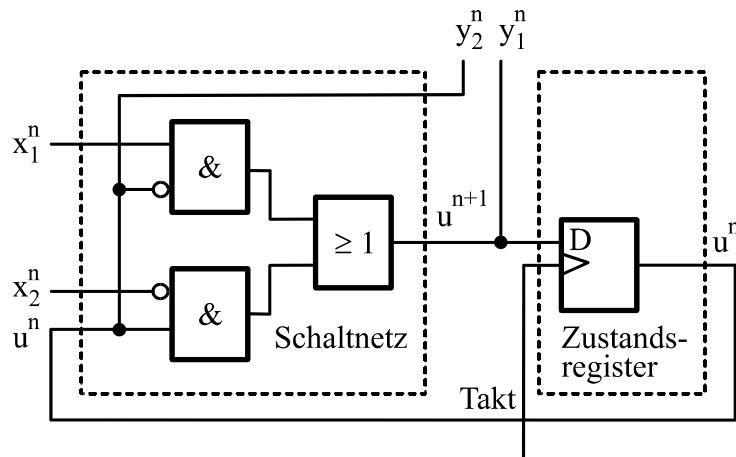


Bild 4.41: Schaltung für extern einstellbares Modulo2-Zählwerk

Ein Vergleich der Übergangsfunktion des Schaltwerkes mit der Übergangsfunktion eines JK-Flip Flops zeigt formale Übereinstimmung:

$$q^{n+1} = \overline{j}q^n \vee \overline{k}q^n$$

Dies legt nahe, eine Schaltungsrealisierung auf der Basis von *JK-Flip Flops* durchzuführen (Bild 4.42).

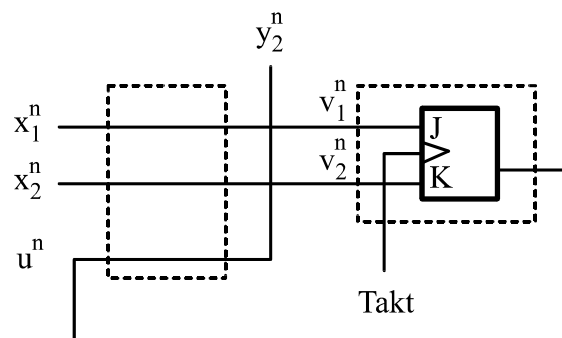


Bild 4.42: Mit JK-FF vereinfachte Schaltung

Allerdings ist zu beachten, dass in dieser Schaltung das Signal $y_1^n = u^n$ nicht zugänglich ist.

Eine Diskussion der Funktionsweise synchroner Schaltwerke zeigt, dass zwar durch die aufgeprägte Taktung des Zustandsspeichers die Fortschaltung des Schaltwerks ohne instabile Zwischenzustände sichergestellt ist. Jedoch können die Ausgangssignale $\underline{y}^n = \{y_1^n, y_2^n, \dots\}$ noch abhängig von Eingangssignaländerungen unkontrolliert wechseln. Abhilfe schafft ein getaktetes Ausgangsregister (Bild 4.43), das die Ausgangssignale taktgesteuert übernimmt.

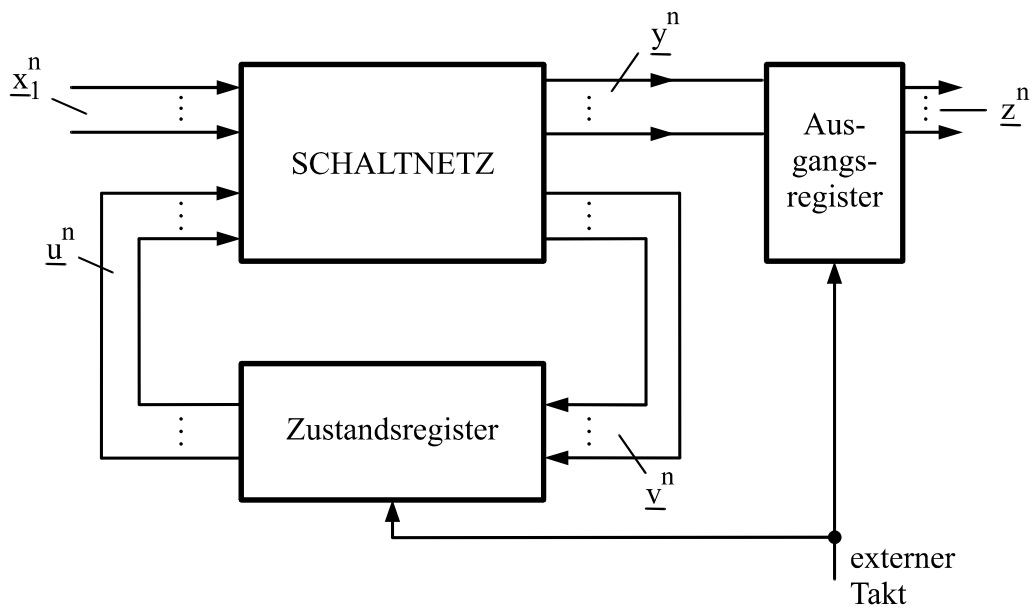


Bild 4.43: Schaltwerk mit getaktetem Ausgangsregister

Bei einem solchen auch im Ausgang getakteten Schaltwerk kann sich der Eingangsvektor außerhalb der aktiven Taktzeit beliebig ändern, Werteänderungen des Zustands- und Ausgangsvektors vollziehen sich stets in gleichen Zeitabständen zu definierten Zeitpunkten. Auf diese Weise kann eine sehr sichere Arbeitsweise, vor allem aber auch ein Gleichlauf mit anderen Schaltwerken erzwungen werden.

4.3.2.3 Ein Modulo4-Steuerwerk

Für den in Kapitel 3 behandelten parallel-seriellen Multiplizierer werde ein Steuerwerk (Bild 4.44) mit folgenden Eigenschaften entworfen:

- fortlaufende Multiplikation von Zahlenpaaren A, B aus 2 synchronen Sequenzen, Zahlen mit jeweils 4 Bit Wortlänge
- Initialisierung und Beendigung durch ein externes Inhibit Signal, dabei nur Ausführung vollständiger Multiplikationen

- SO = 1 parallel laden
- S1 = 1 seriell schieben und der
- Mischzustand SO = 1, S1 = 1 nicht auftritt.

Das zugehörige Steuerwerk ist offenbar ein Modulo 4-Zählwerk, da 4 Bit-Worte verarbeitet werden. Es durchläuft 4 Zustände, wird durch ein Eingangssignal x initialisiert und stillgesetzt. Damit kann ein Zustandsgraph des Steuerwerks bereits entworfen werden, ebenso die Übergangstabelle (Bild 4.46).

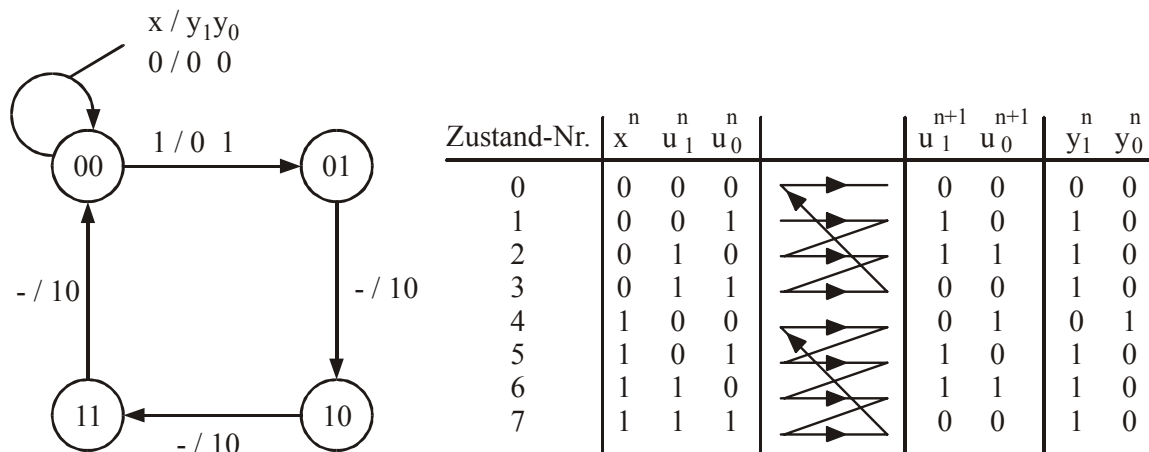
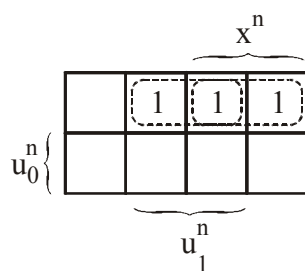


Bild 4.46: Zustandsgraph, Übergangstabelle des Multiplizierer-Steuerwerks

Hieraus folgt die Ausgangs- und Übergangsfunktion (für u_0^{n+1} mit Hilfe eines KV-Diagrammes), siehe Bild 4.47:

u_0^{n+1} :



$$u_0^{n+1} = x^n \overline{u_0^n} \vee u_1^n \overline{u_0^n} = \overline{u_0^n} (x^n \vee u_1^n)$$

$$u_1^{n+1} = \overline{u_1^n} u_0^n \vee u_1^n \overline{u_0^n} = u_1^n \neq u_0^n$$

$$y_0^n = x^n \overline{u_1^n} u_0^n$$

$$y_1^n = u_1^n \vee u_0^n$$

Bild 4.47: KV-Diagramm für das Multiplizierer-Steuerwerk

Das Signal-Zeit-Diagramm (Bild 4.48) zeigt anschaulich die verschiedenen Schaltzustände des Steuerwerkes mit ihren Ausgangssignalen

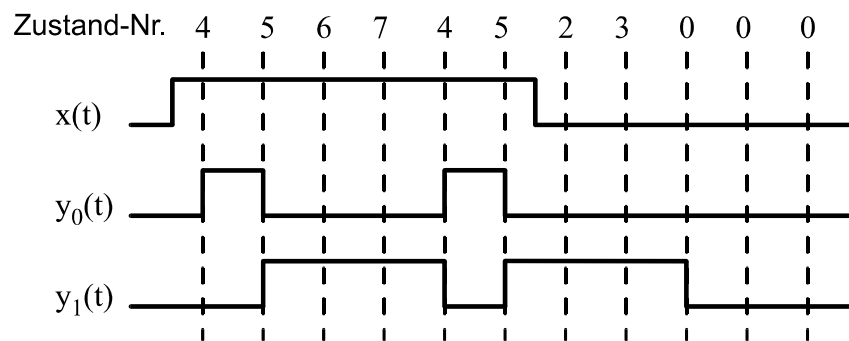


Bild 4.48: Signal-Zeitdiagramm für das Multiplizierer-Steuerwerk

Eine mögliche Schaltungsrealisierung auf der Basis von D-Flip Flops als Zustandsregister zeigt die folgende Skizze (Bild 4.49):

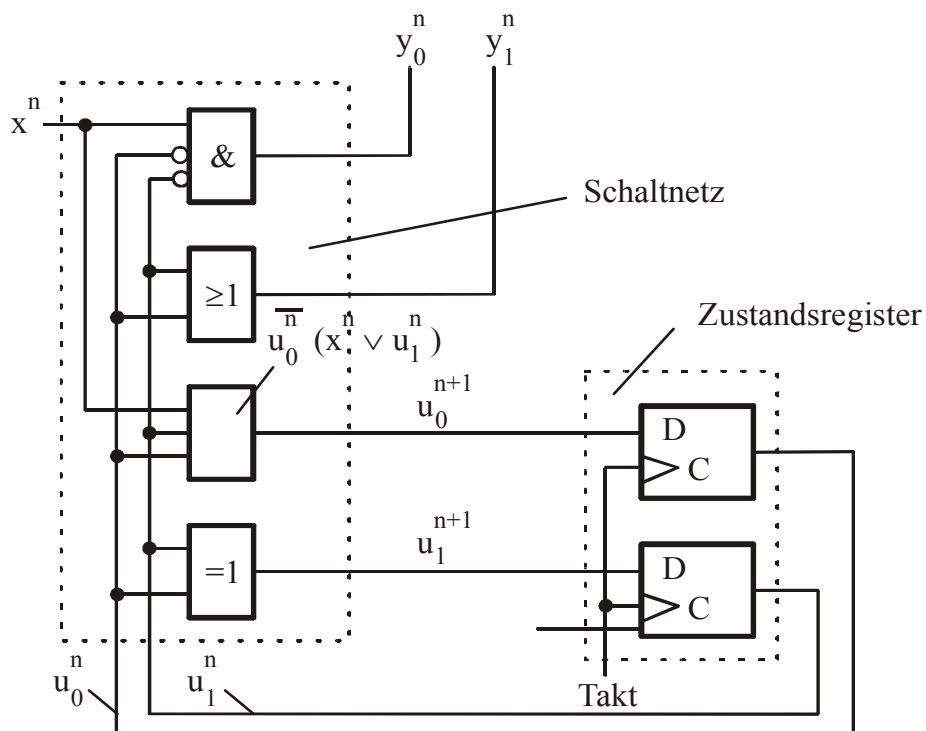


Bild 4.49: Schaltung des Multiplizierer-Steuerwerks

4.3.2.4 Ein Modulo 8-Zählwerk aus Toggle Flip Flops

Ein Modulo n-Zählwerk ist ein autonomes Schaltwerk, dass als n:1 Frequenzteiler eines Taktsignales betrachtet werden kann. Es durchläuft autonom n Zustände. Die folgende Skizze (Bild 4.50) zeigt einen Zustandsgraphen für n=8.

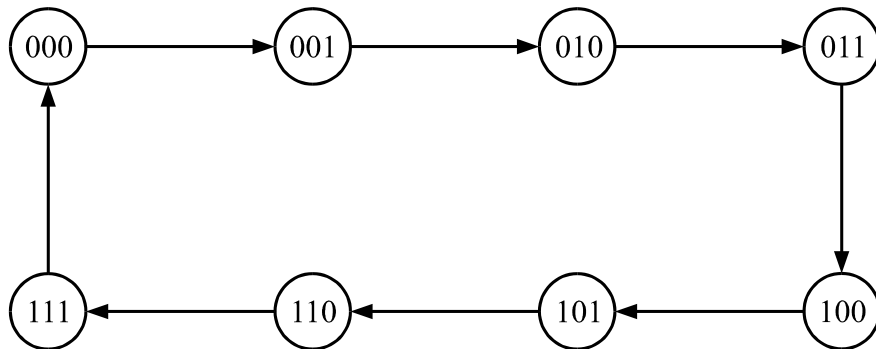


Bild 4.50: Modulo 8-Zählwerk aus Toggle Flip Flops, Zustandsgraph

Aus dem Zustandsgraphen kann ohne weiteres die Übergangstabelle (Tabelle 4.2) angegeben werden. Eingangs- und Ausgangssignale werden nicht berücksichtigt. Die Registerkontrollvariablen v_2^n, v_1^n, v_0^n ergeben sich aufgrund der logischen Gleichung für das Toggle-Flip Flop (Übergangsgleichung):

$$u_i^{n+1} = v_i^n \bar{u}_i^n \vee \bar{v}_i^n u_i^n$$

u_2^n	u_1^n	u_0^n		u_2^{n+1}	u_1^{n+1}	u_0^{n+1}	v_2^n	v_1^n	v_0^n
0	0	0		0	0	1	0	0	1
0	0	1		0	1	0	0	1	1
0	1	0		0	1	1	0	0	1
0	1	1		1	0	0	1	1	1
1	0	0		1	0	1	0	0	1
1	0	1		1	1	0	0	1	1
1	1	0		1	1	1	0	0	1
1	1	1		0	0	0	1	1	1

Tabelle 4.2: Modulo 8-Zählwerk aus Toggle Flip Flops, Übergangstabelle

Die Registerkontrollgleichungen ergeben sich aus der Tabelle zu

$$v_2^n = u_1^n \wedge u_0^n$$

$$v_1^n = u_0^n$$

$$v_0^n = 1$$

Hieraus folgt direkt die Schaltung (Bild 4.51):

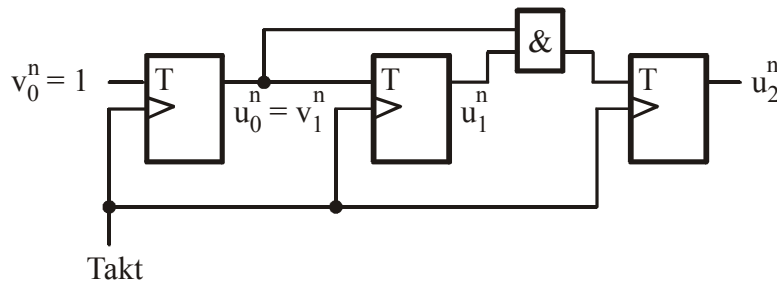


Bild 4.51: Modulo 8-Zählwerk aus Toggle Flip Flops, Schaltung

4.4 Allgemeine Hinweise zum Schaltwerkentwurf

Abschließend zu der Beschreibung der Schaltwerke sollen noch einmal die einzelnen Aufgaben des Schaltwerkentwurfs zusammengestellt werden:

(1) Problemanalyse

Formale Beschreibung der verbal vorgegebenen Eigenschaften des Schaltwerkes zur Definition der Gesamtfunktion einschließlich der Eingangs- und Ausgangsvariablen mit ihrem Zeitverhalten:

- Definition der zu durchlaufenden Zustände
- Angabe der zu den Zuständen gehörigen Zustandsvariablen
- Angabe der Übergänge mit ihren logischen Übergangsbedingungen
- Angabe der zu den Übergängen gehörigen Ausgangsvariablen
- Darstellung des Übergangsverhaltens in Übergangstabelle und Zustandsgraphen

(2) Minimierung der Zustände

Zwei Zustände eines Schaltwerkes bei identischen Eingangsgrößen, demselben Nachfolgezustand und identischen Ausgangsgrößen sind beispielsweise äquivalent. Einer kann fortgelassen werden.

(3) Auswahl der FF-Typen

Offensichtlich führt die Auswahl von D-Flip Flops stets zu einer übersichtlichen (aber oft nicht aufwandoptimalen) Lösung. Häufig können aufgrund der logischen Struktur des Schaltnetzes vorteilhaft alternative Flip Flop-Typen verwendet werden. Hieraus folgt dann die Entwicklung der Registerkontrollgleichungen.

(4) Schaltung

Aus der Entwicklung des Schaltnetzes auf der Basis der Übergangstabelle folgt die endgültige Schaltung des Schaltwerkes.

4.5 Verbindungen zur Automatentheorie nach [LiPP]

Schaltwerke sind technische Realisierungen eines endlichen, diskreten und deterministischen Automaten. Diese wiederum können als mathematische Modelle der Schaltwerke aufgefasst und verwendet werden. Die Motivation der Verwendung von Automaten sei hier kurz dargestellt um eine gedankliche Brücke zwischen Schalt- und Steuerwerken einerseits und der Automatentheorie andererseits darzustellen.

Mit den Schaltnetzen steht eine technische Anordnung zur Verfügung, bei der die Ausgangsgröße zeitlich unmittelbar aus der Eingangsgröße gebildet wird. Hat man umfangreiche Verarbeitungsaufgaben zu realisieren, so bedeutet dies ein sehr großes Eingangsalphabet, das erlauben muss, alle für die Bildung der Ausgangsgröße benötigten Bedingungen als Eingangsgröße darzustellen. Das führt in vielen Fällen zu Schaltnetzen, deren Aufwand technische oder wirtschaftliche Vorgaben oder Möglichkeiten übersteigt.

Man muss daher ein weiteres Verarbeitungskonzept suchen, das die Mächtigkeit des Eingabealphabets reduziert und die Komplexität der Bildung der Ausgabegröße teilweise in eine andere Dimension verlagert. An einem einfachen Beispiel sei dies verdeutlicht: Für die Ausgabe einer Automatenware (z. B. Fahrkarten) sei die Auswahl und Eingabe von vier Elementen aus verschiedenen Alphabeten (z.B. Münzgröße/Scheine, Entfernung, Sondertarif, Kind/Erwachsene) nötig. Fasst man diese Größen schaltnetzmäßig zusammen, so hieße das, entweder die gleichzeitige Betätigung von mindestens vier Tasten bzw. Einwurf-/Einleseoptionen vorzunehmen (was mindestens vier Hände bzw. Fingeraktionen hieße) oder die Schaffung eines Tastensatzes, dessen Mächtigkeit gleich der des kartesischen Produktes aller Mengen ist (was eine Einzeltastenbetätigung erlauben würde). Kartesisches Produkt zweier Mengen $S \times T$: Menge aller geordneten Paare (s, t) mit $s \in S, t \in T$.

Wie wir wissen, wird das Problem auf andere Weise gelöst: das Auswählen und Betätigen erfolgt an solchen Warenautomaten der Reihe nach, d.h. als neue Dimension kommt eine zeitliche Folge der Alphabetzeichen hinzu. Dann und nur dann, wenn die richtige Folge auftrat, wird die angestrebte Ausgangsgröße gebildet, z. B. also die Ware ausgegeben.

Im folgenden sollen diese Überlegungen verallgemeinert werden. Es seien

$$E = \{E_1, E_2, \dots, E_g, \dots, E_u\}$$

ein endliches Eingabealphabet und

$$A = \{A_1, A_2, \dots, A_h, \dots, A_v\}$$

ein endliches Ausgabealphabet. Das neue Konzept lässt sich dann beschreiben als eine Einheit AT, bei der eine zeitliche Folge F_E von Elementen des Eingabealphabets in eine zeitliche Folge F_A von Elementen des Ausgabealphabets abgebildet wird (Bild 4.52).



Bild 4.52: Abbildung der zeitlichen Folge von Elementen des Eingabe- auf zeitliche Folge von Elementen des Ausgabealphabets im Automaten

Damit die Elemente einer Folge in ihrer zeitlichen Reihung unterscheidbar werden, muss ein Ordnungsindex n eingeführt werden.

$$F_E = E_g^n E_g^{n-1} E_g^{n-2} \dots \quad F_A = A_h^n A_h^{n-1} A_h^{n-2} \dots$$

(E_g und A_h sind jeweils beliebig Elemente des Ein- bzw. Ausgabealphabets; ältere Elemente stehen rechts von jüngeren)

Während beim Schaltnetz stets galt

$$A_h^n = f(E_g^n)$$

muß für die neue Einheit offensichtlich gelten

$$A_h^n = f\{E_g^n, E_g^{n-1}, E_g^{n-2}, \dots, E_g^{n-\alpha}\}$$

wenn insgesamt $\alpha+1$ Elemente des Eingabealphabets beobachtet werden müssen, um ein bestimmtes Element des Ausgabealphabets ausgeben zu können. Man nennt α die Zeitliche Tiefe der Verarbeitung ($\alpha \geq 1$; $\alpha = 0$ wäre der Sonderfall des Schaltnetzes).

Die Einheit AT muss also Möglichkeiten aufweisen, α Eingabeelemente aufzubewahren, um zusammen mit dem momentan anliegenden Element E_g^n die Bildung der Ausgabe bewerkstelligen zu können. Man erhält also folgendes Prinzip (Bild 4.53):

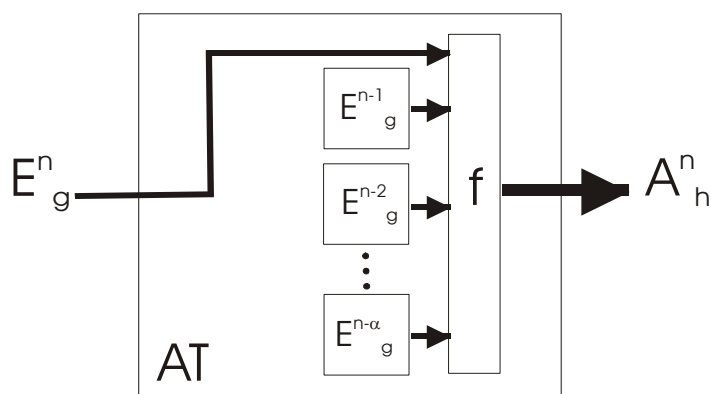


Bild 4.53: Speicherung der Vorgeschichte in einem allgemeinen Automaten

Die Aufbewahrung von α Elementen kann unter Umständen (je nach Mächtigkeit von E und dem Wert von α) recht aufwendig und redundant werden. Es kann daher (in der Regel) sinnvoll sein, die Folge $E_g^{n-1} \dots E_g^{n-\alpha}$ auf ein neues Alphabet $S = \{S_1, \dots, S_k, \dots, S_w\}$ geringerer Mächtigkeit abzubilden:

$$(E_g^{n-1} \dots E_g^{n-\alpha}) \rightarrow S \text{ mit } |E|^\alpha \geq |S|$$

Man nennt das Element $S_k \in S$ Zustand der Einheit AT. Um den Bezug auf den momentan gültigen Wert des Ordnungsindex herzustellen, weist man dem Zustand denselben Index zu. Damit erhält man

$$A_h^n = f(E_g^n, S_k^n)$$

und nennt f die Ausgabefunktion (Ausgangsfunktion) von AT.

Kommt ein neues Eingabeelement hinzu, so muss auf der Grundlage des bisherigen Zustandes (der ja die älteren Elemente von E repräsentiert) das neue E^n zukünftig in einem neuen Zustand, dem Folgezustand, mit berücksichtigt werden.

Es ist also

$$S_k^{n+1} = g(E_g^n, S_k^n)$$

Man nennt g die Überföhrungsfunktion (Zustandsübergangsfunktion). Die Einheit AT kann man also kennzeichnen durch ein Quintupel

$$AT = (E, A, S, g, f)$$

mit drei endlichen Mengen und zwei Abbildungen zwischen diesen Mengen. Man nennt ein solches abstraktes Gebilde einen (endlichen) Automaten.

Ein Automat lässt sich daher grundsätzlich durch folgende Struktur (Bild 4.54) darstellen:

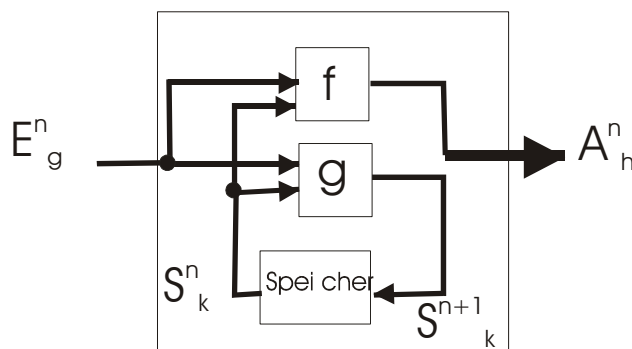


Bild 4.54: Allgemeine rekursive Struktur des Automaten

Man kommt also zu einer rekursiven Anordnung für die Bestimmung des neuen Zustandes. Eine als Speicher bezeichnete Einheit nimmt den momentanen Zustand S^n auf und hält ihn fest. Beim Anlegen eines neuen Eingabeelementes muss der Speicher s^{n+1} übernehmen und den Funktionen g und f als Argument zur Verfügung stellen. Offensichtlich muss dabei der Speicherinhalt zum richtigen Zeitpunkt geändert werden.

In der Literatur sind für Automaten gewisse Typklassen eingeföhrt worden. Im Zusammenhang mit Digitalschaltungen sind dabei Automaten mit den Typmerkmalen endlich, diskret und deterministisch wichtig. Bezöglich der Ausgabefunktion und ihrer Abhängigkeit von E^n und S^n haben drei Fälle einen eigenen Namen erhalten. Es sind der sogenannte **Mealy-Automat** (Bild 4.55) mit

$$A_h^n = f(E_g^n, S_k^n) \text{ als allgemeinstem Fall [Mealy 1955],}$$

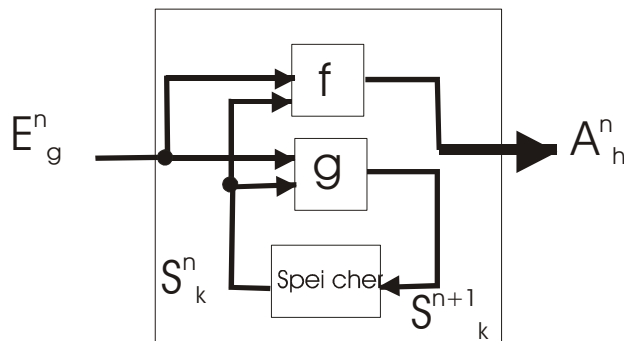


Bild 4.55: Struktur eines Mealy-Automaten

der **Moore-Automat** (Bild 4.56) mit $A = \lambda(S_k^n)$ als dem Spezialfall, bei dem die Ausgabe allein vom Zustand abhängt [Moore 1956],

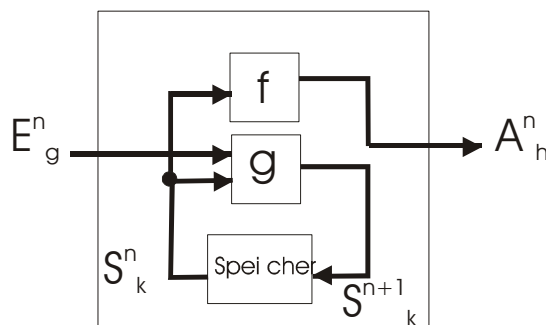


Bild 4.56: Struktur eines Moore-Automaten

und der **Medwedew-Automat** (Bild 4.57) mit $A_h^n \equiv S_k^n$ als der Spezialfall, bei dem als Ausgang der Zustand selbst dient [Gluschkow 1963]:

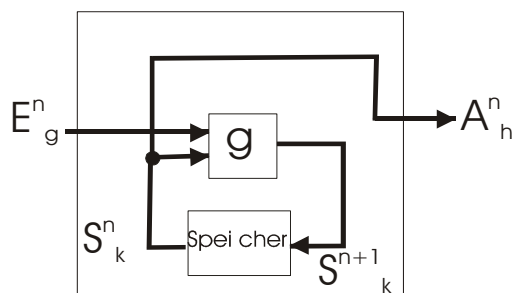


Bild 4.57: Struktur eines Medwedew-Automaten

Zur Festlegung des Verhaltens eines Automaten haben sich vor allem zwei Methoden herausgebildet, der Automatengraph und die Automatentafel.

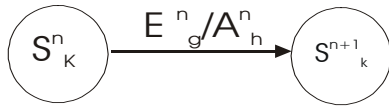
Automatengraph: Die Zustände werden in die Knoten, die Zustandsübergänge in die sie verbindenden, gerichteten Kanten abgebildet. der Graph ist mit Kanten- und beim Moore- wie Medwedew-Automaten auch mit Knotenattributen versehen, die sich auf die Eingabe- bzw. Ausgabeelemente beziehen, s. Bild 4.58.

Beispiel: $E=\{E_1, E_2\}$, $A=\{A_1, A_2, A_3\}$, $S=\{S_1, S_2, S_3, S_4\}$

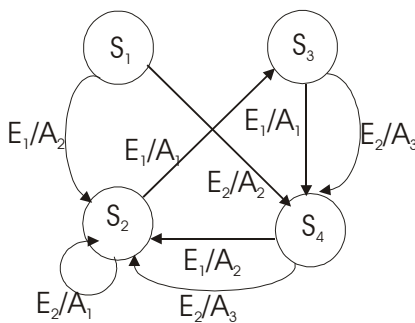
g und f siehe Graph.

Mealy

allgemein:

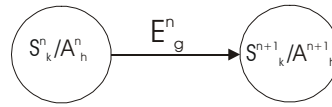


Beispiel:



Moore

allgemein:



Beispiel:

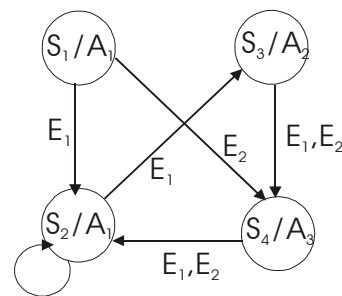


Bild 4.58: Graph eines Mealy- bzw. Moore-Automaten

Automatentafel: Sie entsteht durch Bilden des kartesischen Produktes aus Eingabe- und Zustandsmenge. An den Kreuzungsstellen werden beim Mealy-Automaten der jeweilige Folgezustand und die Ausgabe, beim Moore-Automaten nur der Folgezustand eingetragen; bei diesem wird die Ausgabe zweckmäßigerweise in einer weiteren Spalte notiert.

Allgemein haben Automatentafeln für Moore- bzw. Mealy-Automaten folgendes Aussehen (Tabelle 4.3):

S^n	S^{n+1} / A^n		
	E_1^n	...	E_m^n
.....	...		
S_k^n	...	S_k^{n+1} / A_h^n	...
.....	...		

S^n	S^{n+1}			A^n
	E_1^n	...	E_m^n	
...	...			
S_k^n	...	S_k^{n+1}	...	A_h^n
...	...			

Tabelle 4.3: Tafel eines Mealy- bzw. Moore-Automaten

Folgendes Beispiel zeigt die Tafeln für die gleichen Automaten, die zuvor durch die Automatengraphen dargestellt wurden (Tabelle 4.4):

S^n	S^{n+1} / A^n	
	E_1	E_2
S_1	S_2 / A_2	S_4 / A_2
S_2	S_3 / A_1	S_2 / A_1
S_3	S_4 / A_1	S_4 / A_3
S_4	S_2 / A_2	S_2 / A_3

S^n	S^{n+1}		A^n
	E_1	E_2	
S_1	S_2	S_4	A_1
S_2	S_3	S_2	A_1
S_3	S_4	S_4	A_2
S_4	S_2	S_2	A_3

Tabelle 4.4: Beispiel einer Tafel für einen Mealy- bzw. Moore-Automaten

Beim Übergang vom Automaten zum Schaltwerk müssen die Automatenelemente eindeutig in binäre Größen umgesetzt werden. Im einzelnen bedeutet dies, dass das Eingabealphabet in die Menge der Belegungen des Eingangsvektors, das Ausgabealphabet in die Menge der Belegungen des Ausgangsvektors abgebildet werden müssen. Es ergibt sich:

$$E \leftrightarrow \{X_j\}, \text{ dabei: Mächtigkeit von } E \leq 2^n \text{ mit } \underline{X} = (x_1, \dots, x_n)$$

$$A \leftrightarrow \{Y_j\}, \text{ dabei: Mächtigkeit von } A \leq 2^n \text{ mit } \underline{Y} = (y_1, \dots, y_n)$$

Dieser Vorgang ist mit den Überlegungen bei Schaltnetzen vergleichbar. Neu ist eine dritte Zuordnung, bei der die Zustände S_k der Menge S eindeutig in die Menge der Belegungen eines weiteren Binärvektors, des Zustandsvektors $\underline{U}$ abgebildet werden müssen:

$$S \leftrightarrow \{U_k\} \text{ dabei: Mächtigkeit von } S \leq 2^r \text{ mit } \underline{U} = (u_1, \dots, u_n)$$

Man nennt diesen Vorgang Zustandskodierung. g und f können in Schaltnetze abgebildet werden.

Zur formalen Darstellung der Spezifikation von Schaltwerken können einfach Automatengraph als Schaltwerksgraph und Automatentafel als Schaltwerktafel (oder Tabelle) übernommen werden, indem die Automatenelemente durch ihre Binärrepräsentationen ersetzt werden. für technisch sinnvolle Anwendungen ist diese Darstellungsweise jedoch nicht effizient genug, so daß modifizierte Darstellungen nötig werden.

Bei der Einführung des Zustandbegriffs und der rekursiven Automatenstruktur war lediglich gefordert worden, dass die Indizierung (mit n) eine Ordnung sicherstellt. Bei der Umsetzung in ein Schaltwerk ist dieser Aspekt jedoch technisch zu lösen und zu

überlegen, wie technische Komponenten diese Ordnung sicherstellen können. Dazu ein Gedankenexperiment: wir greifen den Zweig mit Zustandsspeicher, Überföhrungsfunktion und der Rückföhrung heraus, unter der Annahme einer konstanten Eingabe. Bei der Umsetzung in technische Komponenten werden i.a Transistoren die Grundlage der Realisierung bilden, so dass die Schaltung (Bild 4.59) auch verstärkende Elemente enthält.

Bild 4.59: Prinzip der Rückföhrung zur Bildung der Rekursion

Wegen der geschlossenen Rückföhrung kann man die Anordnung (Bild 4.60) mit einem Akustikbeispiel vergleichen, bei dem Mikrofon und Lautsprecher sowohl über einen Verstärker als auch über den Luftschall gekoppelt sein sollen. Wie man aus Erfahrung weiß, kann eine solche Anordnung in einen unkontrollierten Zustand kommen: Es pfeift. Ein Sollbetrieb der Anordnung ist nicht mehr möglich.

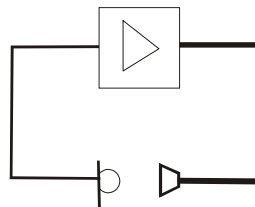


Bild 4.60: Rückgekoppelte akustische Anlage

Aus diesem Beispiel kann man schließen, dass ein rückgekoppeltes Schaltwerk ebenfalls in unzulässige Zustände gelangen kann, die die korrekte Funktion unmöglich machen (vgl. asynchrone Schaltwerke). Man muss daher bei dieser Anordnung beim Entwurf Vorkehrungen gegen dieses Fehlverhalten treffen.

4.6 Prinzip der Mikroprogrammierung

Das Modell des synchronen Steuerwerks mit getrennt realisierter Übergangs- und Ausgangsfunktion bei unveränderbarer Logik zeigt Bild 4.61.

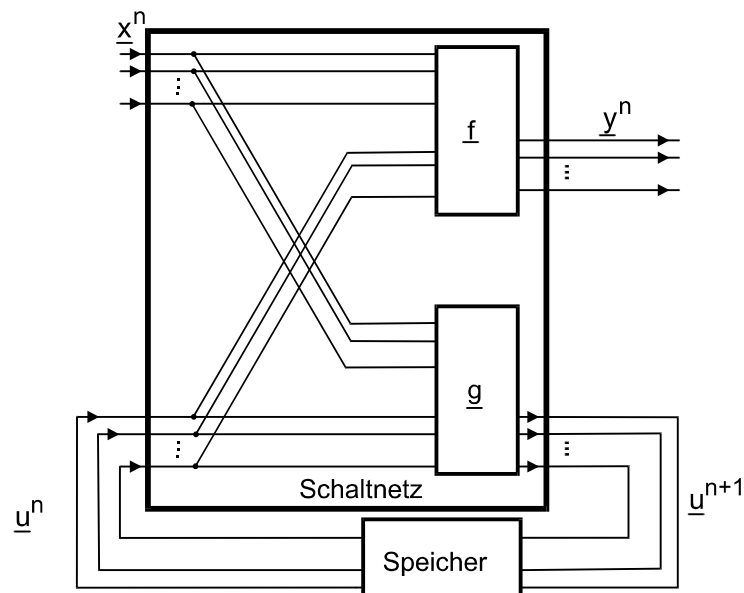


Bild 4.61: Synchrones Steuerwerk mit fester Logik

Das Modell eines mikroprogrammierbaren Steuerwerkes enthält kein festes Schaltnetz aus Logikgattern, sondern programmierbare Festwertspeicher (Bild 4.62).

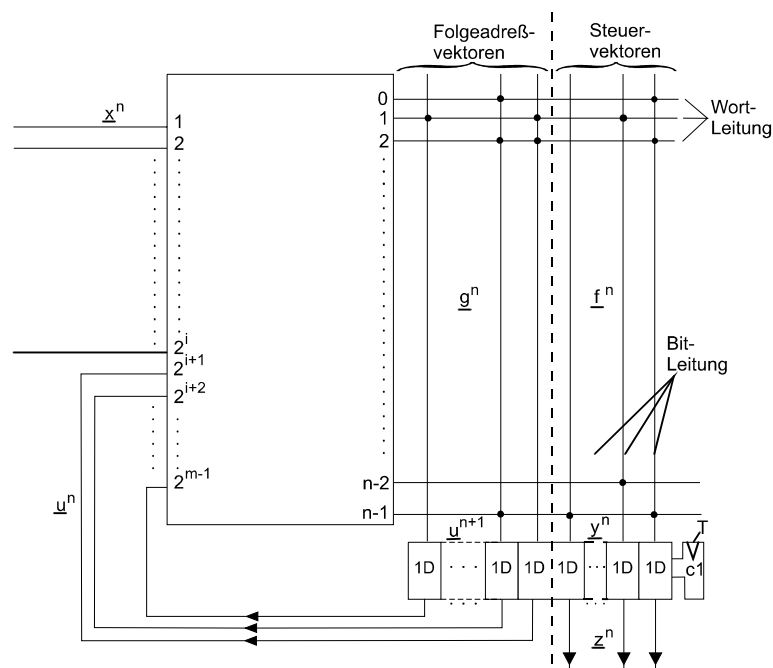


Bild 4.62: Mikroprogrammierbares Steuerwerk

- Eingangs- und Zustandsvariablen (x^n , u^n) bilden die Adresse und aktivieren ein Speicherwort
- Das Speicherwort besteht aus Folgezustandsvektor und Ausgangsvektor $(\underline{u}^{n+1}, \underline{y}^n)$
- Die Speichermatrix enthält somit 2 Bereiche zur Programmierung der Übergangs- und Ausgangsfunktion. Programmierung bedeutet also Einschreiben der Wertetabelle.

Ein Mikroprogrammsteuerwerk besteht aus einem Mikroprogrammspeicher und dem Ausgaberegister. Hieraus ergeben sich äquivalente Benennungen aus der Speicherorganisation, vgl. Bild 4.63:

- Folgezustandsvektor $u^{n+1} \Leftrightarrow$ Folgeadressteil
- Ausgangsvektor $y^n \Leftrightarrow$ Steuerwort

Beide zusammen bilden einen Mikrobefehl:

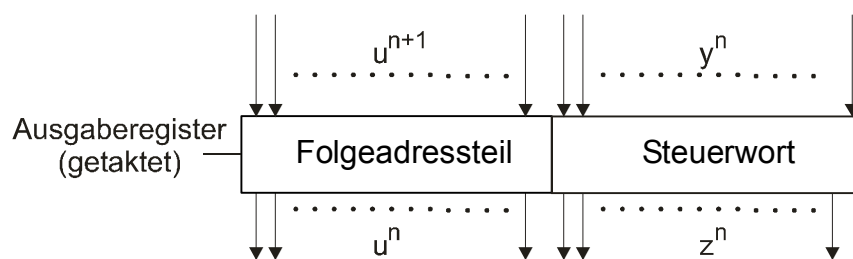
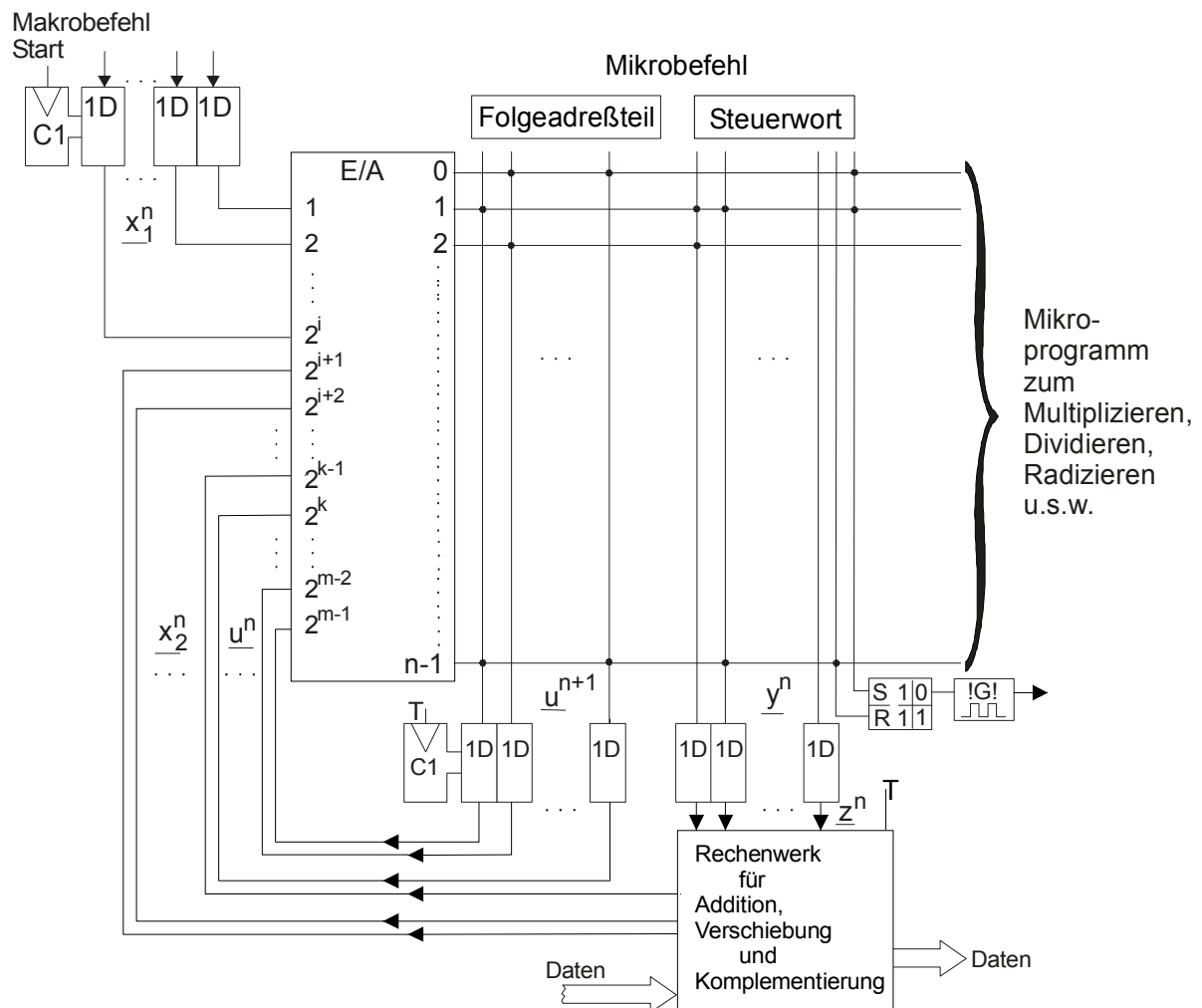


Bild 4.63: Mikrobefehl

- Ein Mikrobefehl steuert eine Operation, die aus einem einzigen Schritt besteht, z.B. eine elementare Hardwareoperation wie Addieren, Verschieben (aber keine Folge von Operationen).
- Das Mikroprogrammsteuerwerk aktiviert eine sequentielle Abarbeitung elementarer Hardwareoperationen.
- Ein Makrobefehl löst ein Mikroprogramm aus, d.h. eine Folge elementarer Operationen.
- Realisierungsparameter sind etwa Speicherbedarf, Taktrate, Steuerworterzeugung, Hardwareaufwand, Flexibilität
- Die Realisierung erfolgt z.B. durch (E)PROM oder RAM

4.6.1 Mikroprogrammsteuerwerk zur Ausführung von Makrobefehlen (Prinzip)



$\underline{x}^n$ – Eingangsvektor $\left\{ \begin{array}{l} \underline{x}_1^n - \text{Makrobefehlsvektor} \\ \underline{x}_2^n - \text{Quittungsvektor} \end{array} \right.$

u^n – Zustandsvektor, u^{n+1} – Folgezustandsvektor

y^n – Ausgangsvektor, z^n – Steuerwortvektor

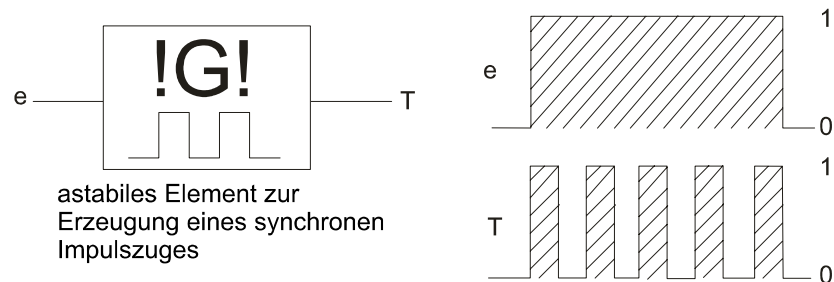
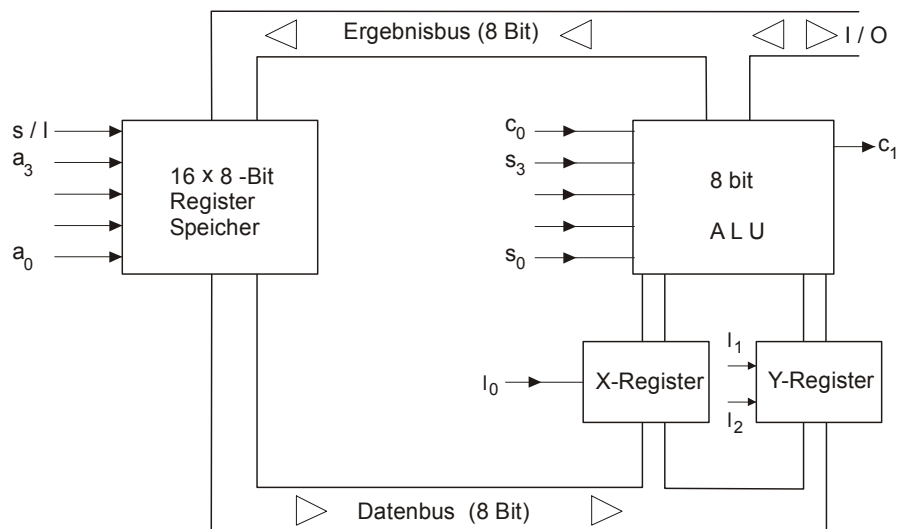


Bild 4.64: Mikroprogrammsteuerwerk mit Rechenwerk

4.6.2 Prinzip einer Register-ALU (RALU) als Bsp. für einfache Rechenwerk-Architektur



- a_0, a_1, a_2, a_3 : Registeradressen zur Adressierung der Operanden x,y
 s/l : Schreib-Lese-Steuerung für Registerspeicher
 s_0, s_1, s_2, s_3 : Steuersignale für Auswahl der ALU-Operation
 c_0 : Übertragseingang für ALU
 c_1 : Übertragsausgang für ALU
 i_0 : Übernahmesignal für X-Register
 i_1, i_2 : Übernahmesignale für Y-Register

Steuerparam.			$s_3 = 1$	$s_3 = 0$	
			logische Operationen	Arithmetrische Operationen	
s_2	s_1	s_0		$c_0 = 0$	$c_0 = 1$
0	0	0	$\underline{z} = \underline{x} \cdot \underline{y}$	$[\underline{z}] = 1$	$[\underline{z}] = 0$
0	0	1	$\underline{z} = \underline{x} \oplus \underline{y}$	$[\underline{z}] = [\underline{y}] - [\underline{x}] - 1$	$[\underline{z}] = [\underline{y}] - [\underline{x}]$
0	1	0	$\underline{z} = \underline{x} \equiv \underline{y}$	$[\underline{z}] = [\underline{x}] - [\underline{y}] - 1$	$[\underline{z}] = [\underline{x}] - [\underline{y}]$
0	1	1	$\underline{z} = \overline{\underline{x}} \cdot \underline{y}$	$[\underline{z}] = [\underline{x}] + y$	$[\underline{z}] = [\underline{x}] + [\underline{y}] + 1$
1	0	0	$\underline{z} = \underline{x} \cdot \overline{\underline{y}}$	$[\underline{z}] = [\underline{y}]$	$[\underline{z}] = [\underline{y}] + 1$
1	0	1	$\underline{z} = \underline{x} + \underline{y}$	$[\underline{z}] = [\underline{y}]$	$[\underline{z}] = [\underline{y}] + 1$
1	1	0	$\underline{z} = \overline{\underline{x}} + \underline{y}$	$[\underline{z}] = [\underline{x}]$	$[\underline{z}] = [\underline{x}] + 1$
1	1	1	$\underline{z} = \underline{x} + \overline{\underline{y}}$	$[\underline{z}] = [\underline{x}]$	$[\underline{z}] = [\underline{x}] + 1$

Bild 4.65: Register-ALU und Tabelle der einstellbaren ALU-Funktionen

4.6.3 Mikroprogrammierbare Rechner (Firmware)

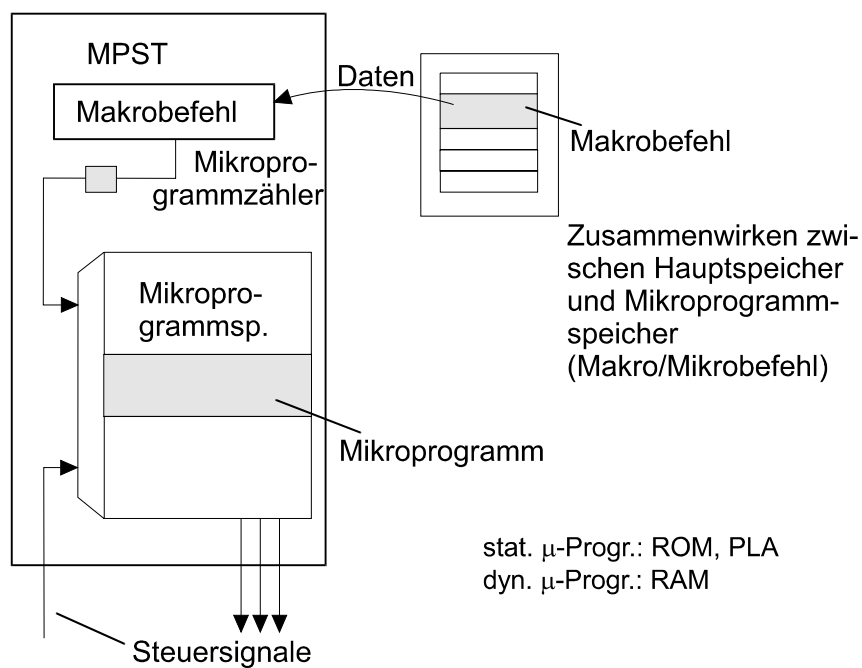
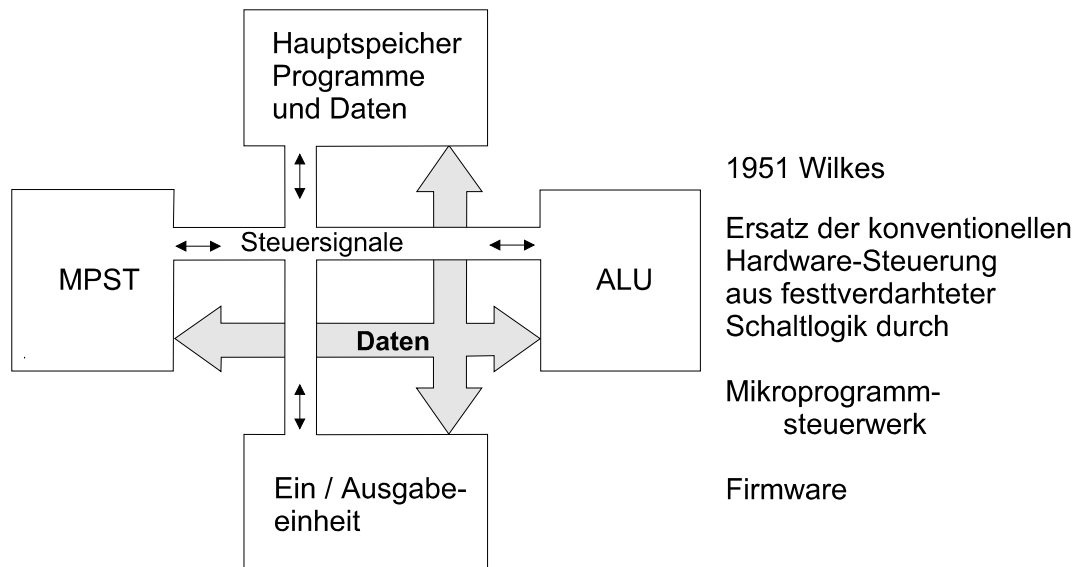


Bild 4.67: Mikroprogrammierung im Rechner

Vorteile:

- erhöhte Flexibilität, Veränderbarkeit des Befehlssatzes eines Rechners
- flexibel änderbare Datenflusssteuerung zwischen den Rechnerkomponenten
- Ergänzung bestehender Systeme durch neue Makrobefehle
- Emulation anderer Rechnerstrukturen

Die Bedeutung der Mikroprogrammierung in Rechnern hat erheblich abgenommen. Gründe hierfür sind:

- Das Fehlen von Anwendungssoftware, denn die Einführung neuer Mikrobefehle verlangt deren Einsatz in Anwendungssoftware. Ein Nutzer muss diese dann anpassen.
- Mikroprogrammierung ist typisches Kennzeichen von CISC-Prozessoren (Complex Instruction Set Computer). Moderne RISC-Rechner (Reduced Instruction Set Computer) vermeiden MP aus Effizienzgründen und verwenden andere Prinzipien zur Effizienzsteigerung (s. weiterführende Vorlesungen)

Kapitel 5

Speicherstrukturen und logische Felder

5.1 Halbleiterspeicher

RAM (**R**andom **A**ccess **M**emory): Schreib-Lese-Speicher

SRAM: statisches RAM

DRAM: dynamisches RAM

ROM (**R**ead **O**nly **M**emory): Festwertspeicher

ROM: maskenprogr. ROM

PLA: maskenprogr. PLA (**P**rogrammable **L**ogic **A**rray – progr. logisches Feld)

PROM: kundenprogramm. ROM

FPLA: kundenprogramm. PLA (F: Field)

PAL: kundenprogramm. UND-Matrix (**P**rogrammable **A**rray **L**ogic)

EPROM: löschbares PROM (E: erasable)

EEPROM: elektr. löschbares PROM

Speicherbausteine werden völlig anwendungsunabhängig gefertigt. Ihre Programmierung erfolgt jeweils durch den Benutzer. Zur Programmierung muß der Inhalt der Speicherbausteine festgelegt werden. Bei den verschiedenen Speichertypen erfolgt dies in unterschiedlicher Form.

- **ROMs** (**R**ead **O**nly **M**emories) können mittels der Fertigungsmasken beim Hersteller programmiert werden, so daß es sich im eigentlichen Sinne bei ROMs nicht um anwenderprogrammierbare Bausteine handelt)
- bei **PROMs** (**P**rogrammable **R**ead **O**nly **M**emories) werden in den Zellen vom Anwender selektiv Sicherungen durchgebrannt (dies ist ein irreversibler Vorgang)
- **EPROMs** (**E**rasable and **P**rogrammable **R**ead **O**nly **M**emories) können mittels UV-Bestrahlung wieder gelöscht werden

- **EEPROMs** (**E**lectric **E**rasable and **P**rogrammable **R**ead **O**nly **M**emories) können elektrisch beschrieben und gelöscht werden
- **RAMs** (**R**andom **A**ccess **M**emories) können während des Normalbetriebs verändert werden, ihr Inhalt geht während des Abschaltens der Versorgungsspannung verloren.

Die allgemeine Struktur eines Speicherblockes ist in Bild 5.1 dargestellt. Diese Struktur ist bis auf Unterschiede bei den Schreib-/ Leseverstärkern für RAMs und ROMs identisch.

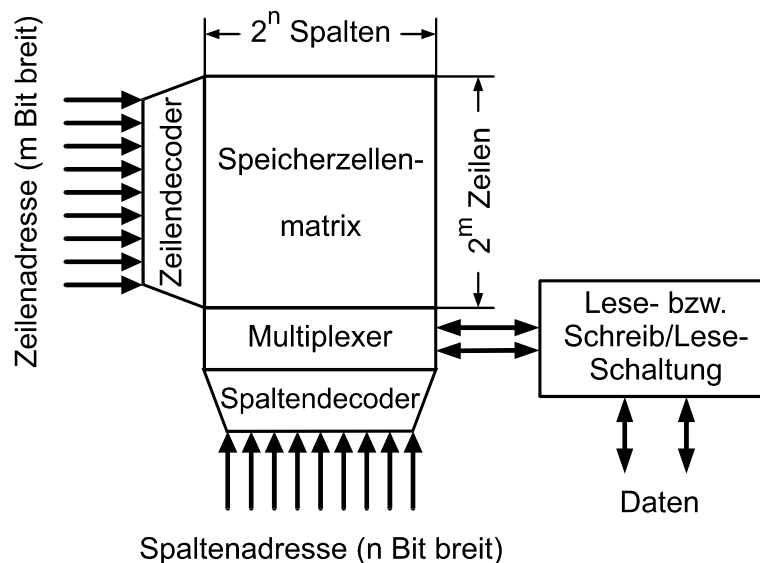


Bild 5.1: Allgemeine Struktur eines Speicherblockes

ROMs können innerhalb integrierter Schaltungen als maskenprogrammierbare Blöcke entworfen werden. Maskenprogrammierbar bedeutet in diesem Zusammenhang, daß die im Speicher enthaltene Information bereits in den Masken des Chipfertigungsprozesses enthalten ist und nicht durch nachträgliche Programmierung eingebracht wird. ROMs können von Blockgeneratoren erzeugt werden.

Die einzelnen Speicherzellen werden rechteckig in einer Speicherzellenmatrix angeordnet. Diese Matrix enthält Zeilen und Spalten, die über Decoder ausgewählt werden können. Die Adresse der zu adressierenden Speicherzelle wird in eine Zeilen- und Spaltenadresse aufgeteilt. Die Leitungen, die vom Zeilendecoder in die Matrix führen, werden als Wortleitungen bezeichnet, und diejenigen, die vertikal aus der Matrix herausführen, werden als Bitleitungen bezeichnet. Bei m Bits für die Zeilenadresse selektiert der Zeilendecoder eine von 2^m Zeilen. Alle Inhalte der adressierten Zeile stehen somit am Multiplexer an. Da dieser von einem Spaltendecoder angesteuert wird, kann mit Hilfe der n Bits der Spaltenadresse eine

von 2^n Spalten ausgewählt werden. Der Multiplexer schaltet somit die entsprechende Spalte zum Leseverstärker durch. Der adressierte Speicherzelleninhalt steht dann am Ausgang der Schaltung zur Verfügung.

Die Speicherzellenmatrix eines ROM-Speichers hat einen einfachen Aufbau. In Bild 5.2 ist ein Ausschnitt aus einer solchen Speichermatrix dargestellt.

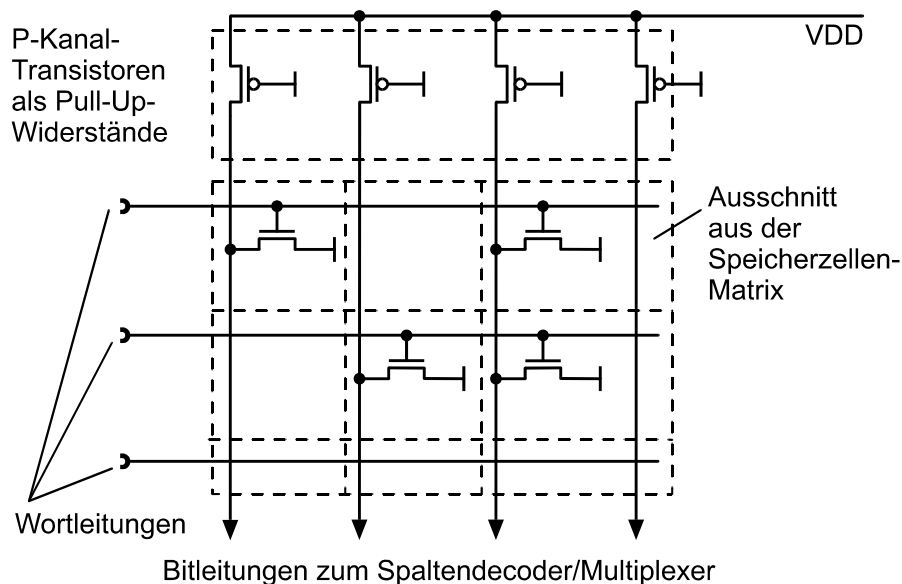


Bild 5.2: Aufbau einer Speicherzellenmatrix eines ROM-Speichers

Die einzelnen Speicherzellen enthalten je nach gespeichertem Wert entweder einen n-Kanal-Transistor oder keinen Transistor. Das Gate des n-Kanal-Transistors ist mit der zugehörigen Wortleitung verbunden, der Source-Anschluß mit der Bitleitung und Drain mit Masse. Die senkrechten Leitungen zum Spaltendecoder (die Bitleitungen) werden von als Lastelementen beschalteten p-Kanal-Transistoren auf VDD-Potential gezogen. Bei der Selektion einer Zelle wird die zugehörige Wortleitung auf VDD-Potential gebracht. Wenn in einer Speicherzelle ein Transistor vorhanden ist, schaltet dieser durch und zieht die entsprechende Bitleitung auf GND-Potential. Ein in einer Matrixzeile vorhandener Transistor bedeutet damit eine gespeicherte "0", ein nicht vorhandener eine "1". Je nach zu programmierendem logischen Wert setzt ein ROM-Generator beim Entwurf eines ROM-Arrays also einen Transistor in eine Matrixzelle oder nicht.

Weitere Formen von ROMs ergeben sich dadurch, daß ROMs statisch und dynamisch implementiert werden können. Das Prinzip der matrixförmigen Zellanordnung bleibt dabei jedoch erhalten. Einen Überblick zu weiteren Implementierungsformen von ROMs gibt z.B. [Reifsch98].

RAMs werden mit Hilfe der gleichen, in Bild 5.1 dargestellten Grundstrukturen aufgebaut wie ROMs. Da die Speicherinhalte dieser Zellen gelesen und geschrieben werden können, muß die Leseschaltung ersetzt werden durch eine Schreib-/ Lese-Schaltung. Die Speicherzellen selbst sind daher auch komplexer aufgebaut als die eines ROMs. Wie die ROMs können aber auch die RAMs von Blockgeneratoren nach Vorgabe der Speichergröße und der Organisation automatisch erzeugt werden.

Man unterteilt die RAMs noch in *DRAMs* und *SRAMs* (dynamische und statische RAMs). SRAMs sind statische Speicherelemente, die ihren Inhalt bei Anlegen der Versorgungsspannung beliebig lange halten können (z.B. 6-Transistor-Zelle). DRAMs sind dynamische Speicherelemente (z.B. 1-3 Transistor-Zellen), die eine wesentlich höhere Integrationsdichte erlauben, die ihren Inhalt aber nur eine gewisse Zeit lang halten können. Sie erfordern ein periodisches Wiederauffrischen (*Refresh*) der auf Kapazitäten gehaltenen Ladung (Information).

Bild 5.3 stellt eine statische 6-Transistor RAM-Zelle einer dynamischen 3-Transistor RAM-Zelle (jeweils in CMOS-Technologie) gegenüber.

Die in Bild 5.3 a) dargestellte 6-Transistor-Zelle ist aus vier n-Kanal- und zwei p-Kanal-Transistoren aufgebaut. Die beiden n-Kanal-Transistoren T_3 und T_5 bilden dabei zusammen mit den p-Kanal-Transistoren eine bistabile Kippstufe. Liegt einer der beiden Ausgangsknoten auf VDD-Potential, so ist der darüber liegende p-Kanal-Transistor durchgeschaltet. Der jeweils andere Ausgangsknoten, der ja mit dem Gate des p-Kanal-Transistors verbunden ist, liegt dann auf GND-Potential. Die Zelle ist so aufgebaut, daß im statischen Fall keine Leistung verbraucht wird, da kein Strom von VDD nach GND fließt. Diese 6-Transistor-Zelle kann eine einmal eingeschriebene Information beliebig lange halten. Ein weiterer Vorteil dieser Zelle ist, daß keine aufwendigen Leseschaltungen erforderlich sind (siehe z.B. [Reifsch98]).

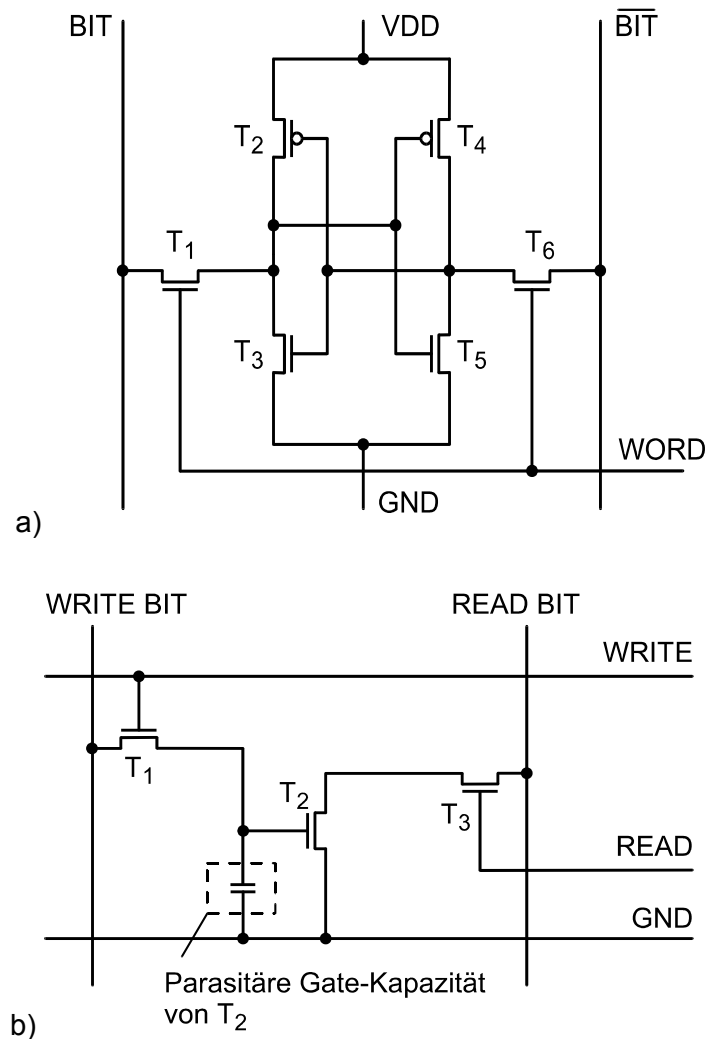


Bild 5.3: a) Statische 6-Transistor RAM-Zelle, b) dynamische 3-Transistor RAM-Zelle

Wesentlich platzsparender lassen sich dynamische RAM-Zellen aufbauen. Bild 5.3 b) zeigt zum Vergleich den Aufbau einer 3-Transistor DRAM-Zelle. Bei diesen dynamischen Zellen wird die Information nur in Kapazitäten gehalten. Da parasitäre Leckströme diese Kapazitäten mit der Zeit entladen, ist ein periodischer Refresh der Zelleninhalte notwendig. Bei der 3-Transistor DRAM-Zelle ist je eine Bitleitung zum Schreiben und zum Lesen der Information vorhanden. Die Information wird dabei auf der parasitären Gate-Kapazität des Speichertransistors T_2 gehalten.

Die Realisierung von Speichern mit noch höherer Dichte ist mit Hilfe von 1-Transistor DRAM-Zellen möglich, siehe hierzu z.B. [Reifsch98].

Insgesamt sind dynamische Speicherzellen flächensparender zu realisieren als statische Speicherzellen. Der bei dynamischen RAMs notwendige periodische Refresh erfordert allerdings einen höheren Ansteuerungsaufwand als bei statischen RAMs. Generell kann gesagt werden, daß dynamische RAM-Speicher schwieriger zu dimensionieren sind als statische. Bei jedem Entwurf gilt es daher abzuwägen, ob

ein flächenoptimierter Speicher auf der Basis von z.B. dynamischen 3-Transistor-Zellen entworfen werden soll (wie z.B. für on-board caches bei schnellen ASIC-Anwendungen) oder eher ein leichter zu dimensionierender statischer Speicher.

5.1.1 Speicher mit bitweiser Organisation, Speicher-Adressierung durch Koinzidenz

Die gesamte Adresse bilden M Bit, die z. B. für eine quadratische Speichermatrix je zur Hälfte einem Spalten- bzw. einem Reihen-Adressdecoder zugeführt werden vgl. Bild 5.1. Von dort werden die einzelnen Speicherzellen adressiert und können über den Datenausgang mit Enable-Steuerung ausgelesen werden (Bild 5.4).

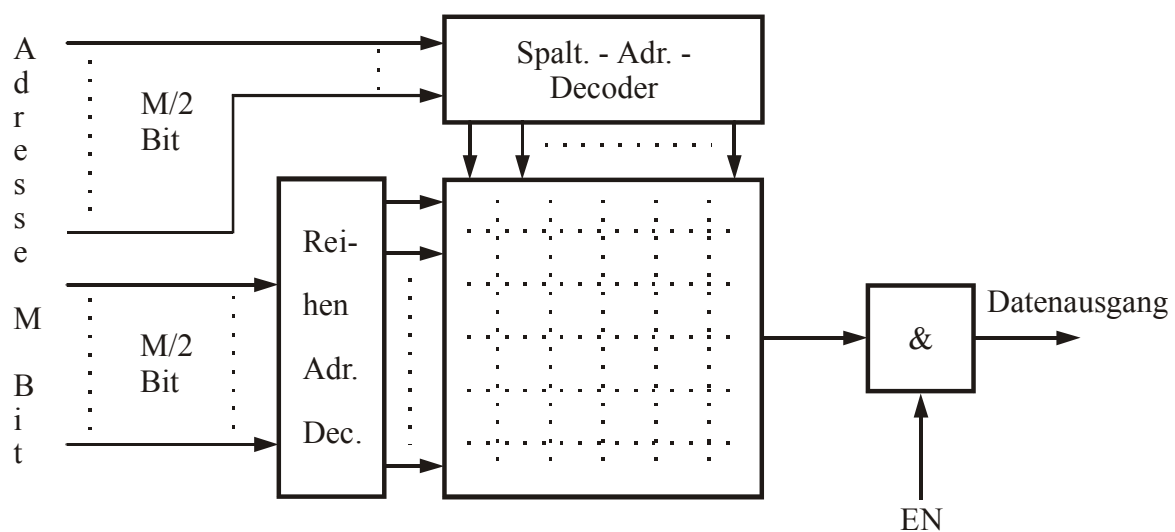


Bild 5.4: Koinzidenz-Adressierung

5.1.2 Speicher mit wortweiser Organisation, ein- / zweidimensionale Adressierung

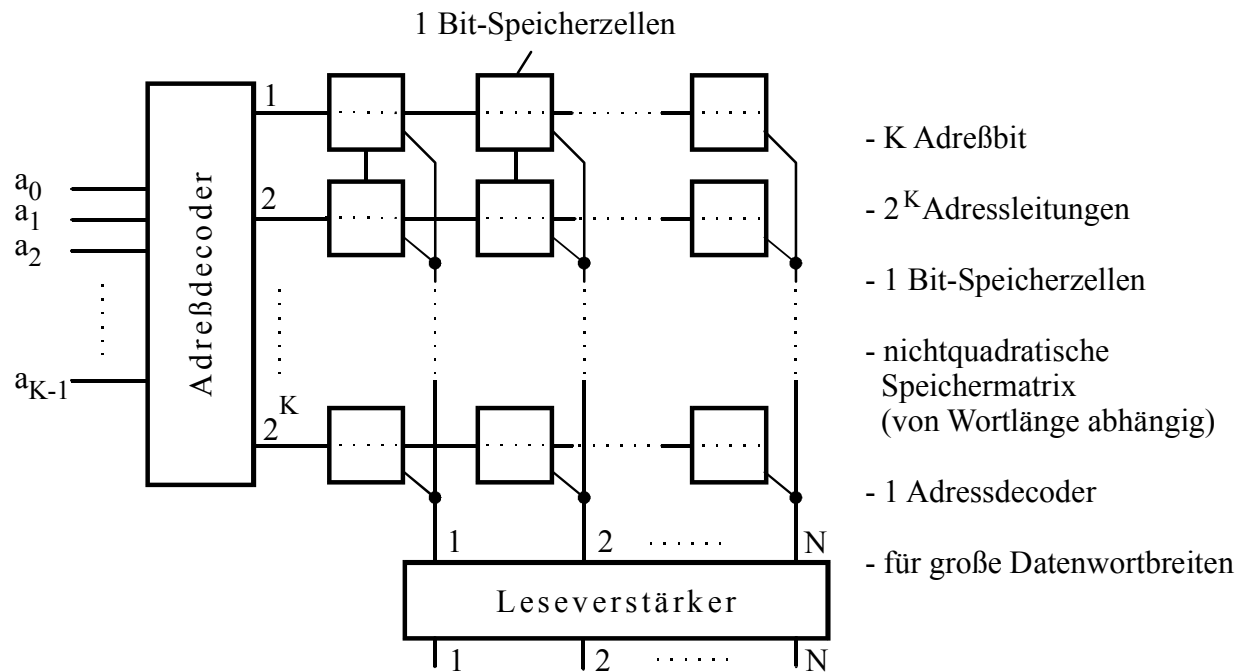


Bild 5.5: Eindimensionale Adressierung

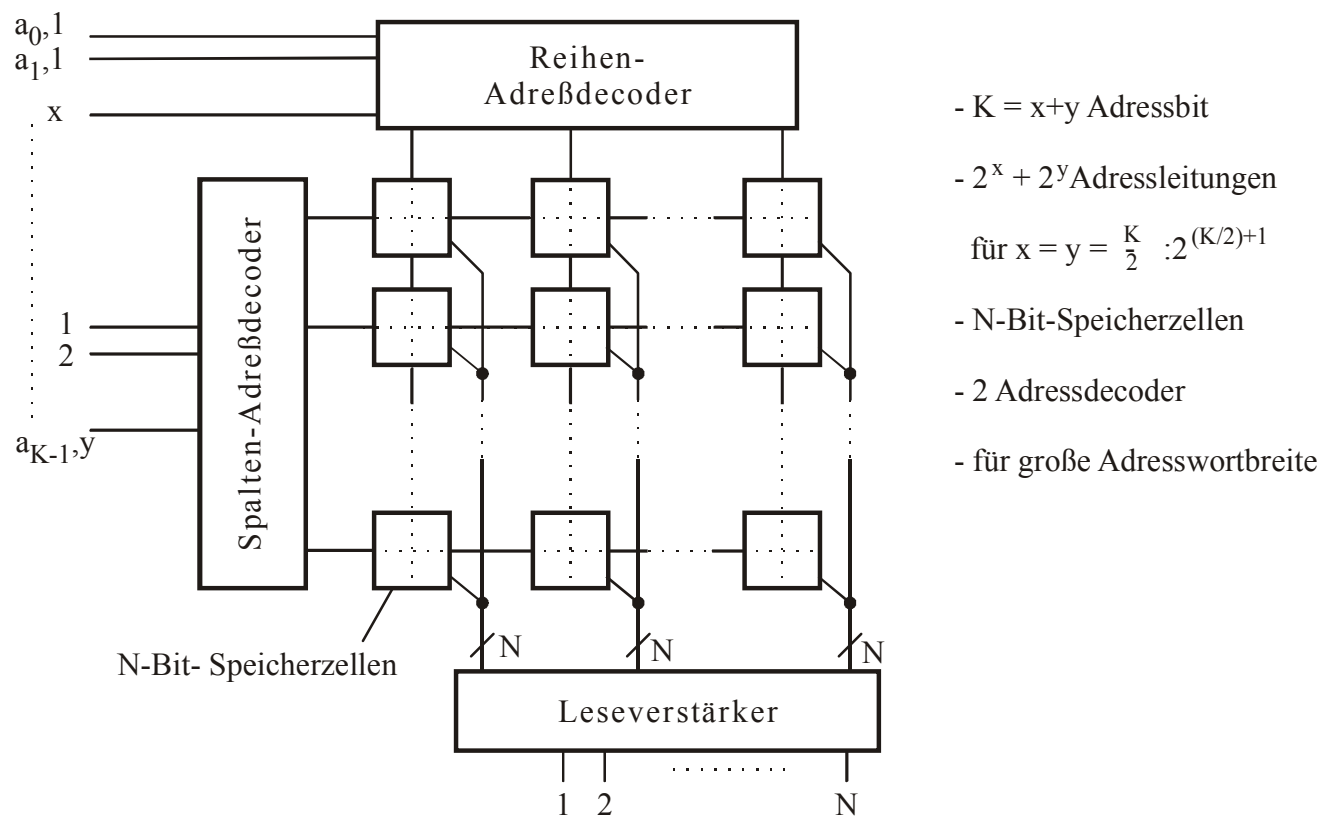


Bild 5.6: Zweidimensionale Adressierung

5.1.3 Adressdecoder zur Speicheradressierung

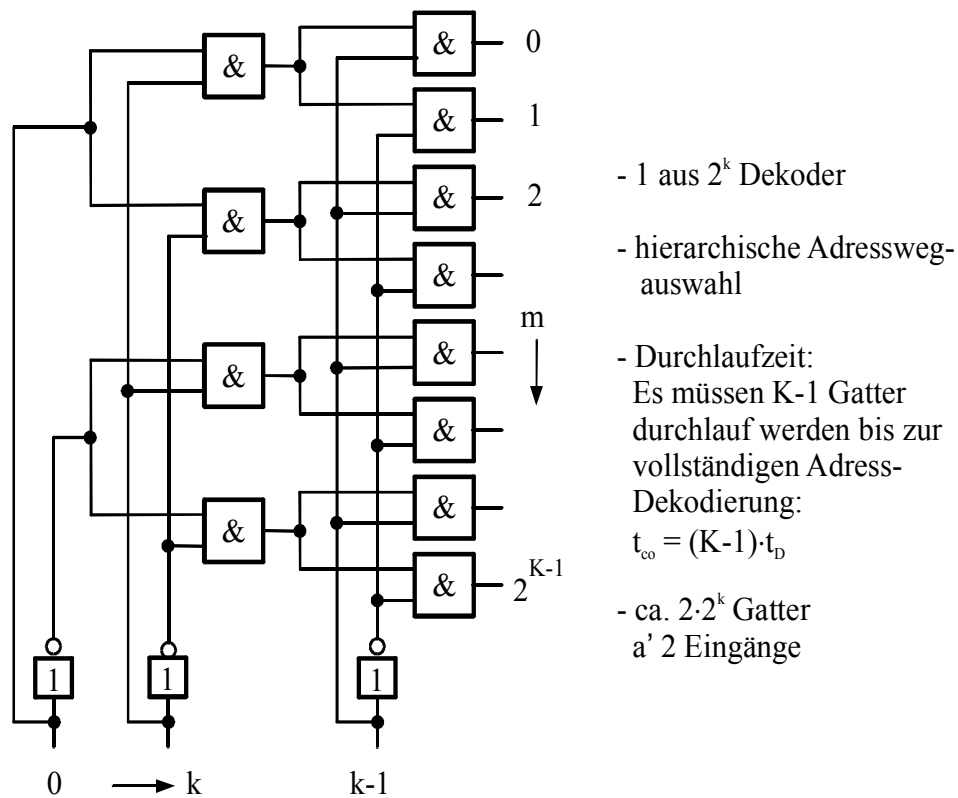


Bild 5.7: Schalterbaum-Dekodierung

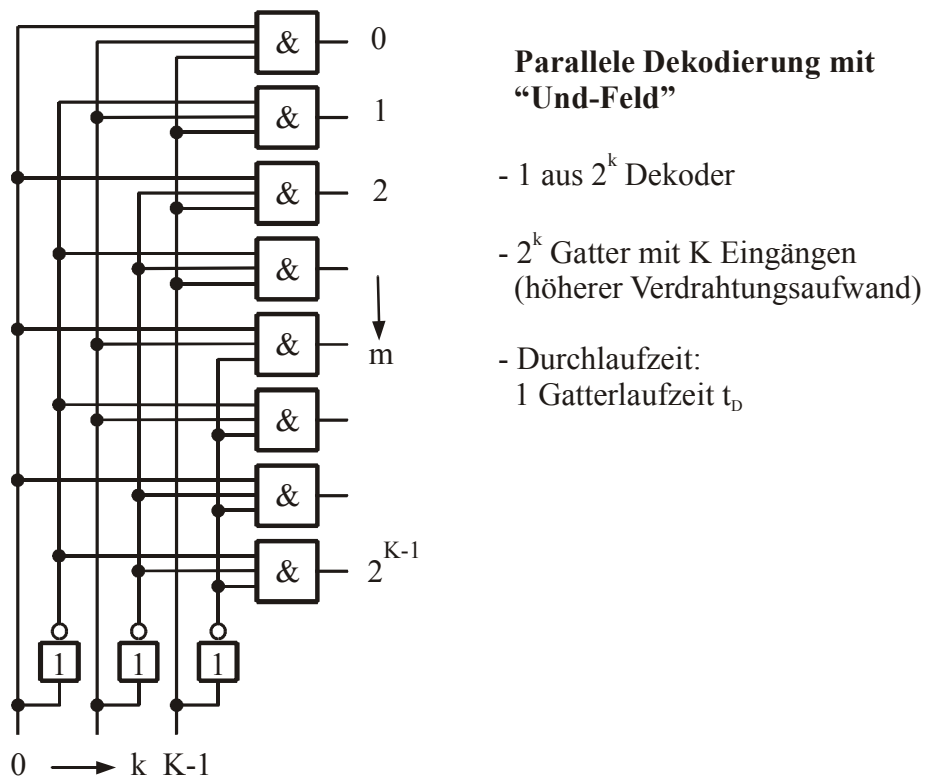


Bild 5.8: UND-Feld-Dekodierung

5.2 Logische Beschreibung von Festwertspeichern

Betrachtet man die Funktion eines Festwertspeichers ohne sich weiter um die innere Organisation zu kümmern, so erkennt man, daß über einen Adressdecoder ein Datenspeicherblock ausgewählt wird, dessen Datenwort dann im Ausgangsregister zur Verfügung gestellt wird (Bild 5.9).



Bild 5.9: Logische Beschreibung von Festwertspeichern

Der Adressdecoder läßt sich dabei als "1 aus 2^K "-Decoder auffassen. Bei diesem wird jeder Kombination (von 2^K möglichen) der M Adressbits ein Ausgangsbit zugeordnet und es wird mit diesem Bit über eine einzelne Adreßleitung der gesuchte Block ausgewählt.

Die Adressberechnung entspricht einer Wahrheitstabelle, bei der jedem Zustand eine der Eingangsvariablen eine Ausgangsvariable zugeordnet ist und kann entsprechend den folgenden Gleichungen realisiert sein. Jede logische Gleichung repräsentiert dabei einen Minterm der Eingangsvariablen. Es sind in der Adressberechnung alle Minterme möglich.

$$\begin{aligned}
 H_0 &= \bar{h}_{00} \wedge \bar{h}_{10} \wedge \bar{h}_{20} \dots \bar{h}_{(K-1)0} \\
 H_1 &= h_{01} \wedge \bar{h}_{11} \wedge \bar{h}_{21} \dots \bar{h}_{(K-1)1} \\
 H_2 &= h_{02} \wedge h_{12} \wedge \bar{h}_{22} \dots \bar{h}_{(K-1)2} \\
 &\vdots \\
 H_{M-1} &= h_{0(M-1)} \wedge h_{1(M-1)} \wedge h_{2(M-1)} \dots h_{(K-1)(M-1)}
 \end{aligned}$$

Somit realisiert ein Festwertspeicher eine Wahrheitstabelle mit $K=M$ Eingangsvariablen und N Ausgangsvariablen (siehe Bild 5.10).

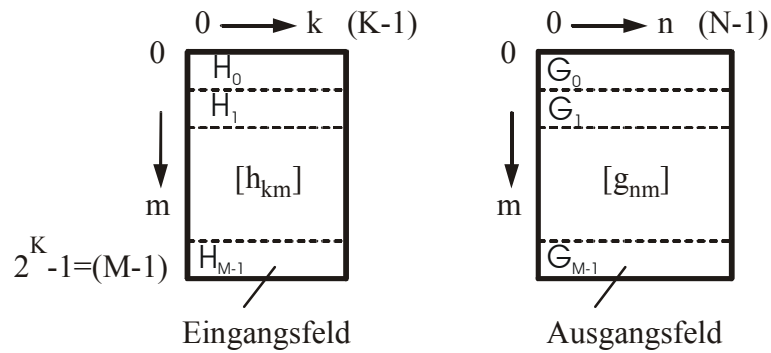


Bild 5.10: Eingangs- und Ausgangsfeld zur Festwertspeicherbeschreibung

mit M Eingangsvektoren $\underline{H}_m$ im Eingangsfeld

$$\underline{H}_m = h_{0m}, h_{1m}, \dots, h_{(K-1)m}$$

jeder Vektor mit

$$K = \text{Id } M$$

Komponenten. Im Ausgangsfeld sind ebenfalls M Vektoren $\underline{G}_m$ definierbar mit jeweils N Komponenten entsprechend der Ausgangsdatenwortlänge:

$$\underline{G}_m = g_{0m}, g_{1m}, \dots, g_{(N-1)m}$$

Hieraus lassen sich die Ausgangsvektoren aus den Eingangsvariablen berechnen (Funktionsbündel)

$$\underline{G}_m = f(\underline{H}_m)$$

Aus diesen Betrachtungen lässt sich die im Bild 5.11 dargestellte PROM-Array-Struktur entwickeln.

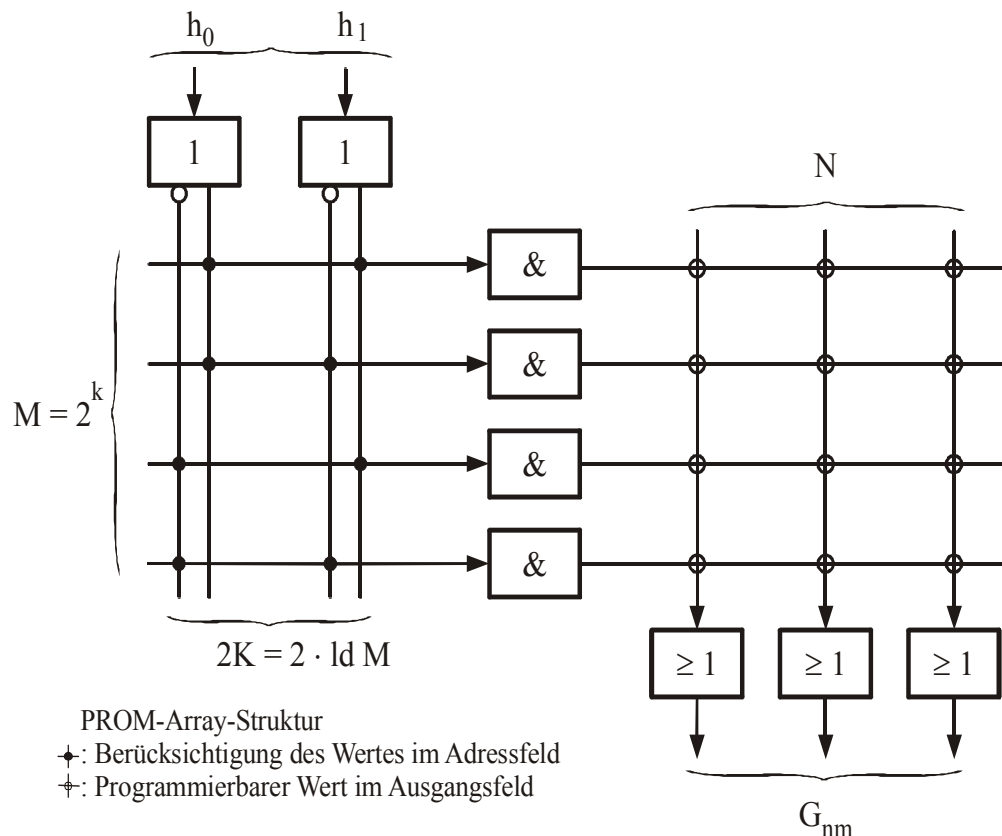


Bild 5.11: PROM-Blockschaltbild

Mit Hilfe dieser Gleichungen lassen sich die Komponenten n -ter Ordnung der M Ausgangsvektoren berechnen. Mit Hilfe der disjunktiven Normalform für die Komponente $n=i$ ergibt sich z. B.

$$g_i = (\bar{h}_{00} \wedge \bar{h}_{10} \wedge \dots \wedge h_{(K-1)0}) \vee \dots \\ \vee (h_{01} \wedge h_{11} \wedge \dots \wedge h_{(K-1)1}) \vee \dots$$

Bei dieser Darstellung ist für jeden von Null verschiedenen Bitwert im Ausgangsfeld ein (in Klammern gesetzter) Konjunktionsterm zu berücksichtigen. Dabei entspricht jedem Bit im Ausgangsfeld ein programmierbarer Kreuzungspunkt im Ausgangsfeld des PROM's ("Oder-Array").

5.3 Programmierbare logische Felder

In vielen Anwendungsfällen sind jedoch nicht alle möglichen Kombinationen der Eingangsvariablen erforderlich. Dies ist etwa dann der Fall, wenn das Ausgangsfeld stark mit Nullen besetzt ist. Dann ist die Bereitstellung einer vollständigen Wahrheitstabelle mit dem Festwertspeicher ein zu großer Aufwand. In diesem Fall lässt sich die Wertetafel günstig durch PAL bzw. PLA-Schaltkreise realisieren. Die folgende Skizze zeigt eine typische PLA-Architektur.

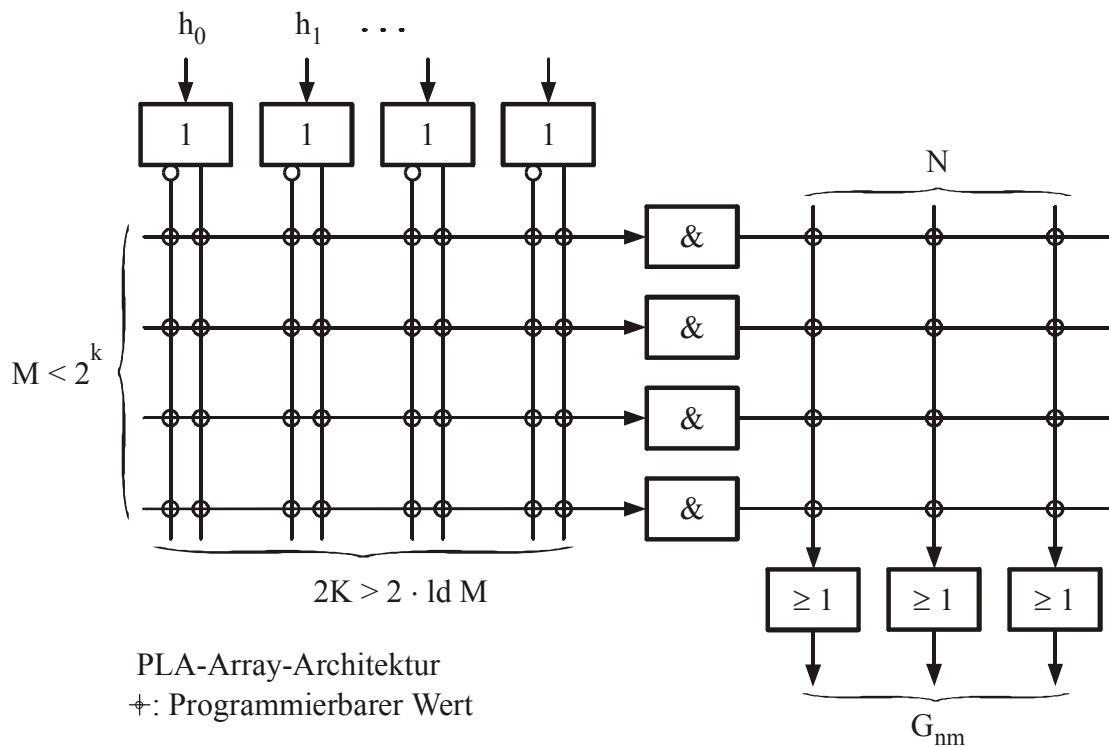


Bild 5.12: PLA-Blockschaltbild

Die beiden sind bezüglich der UND- und ODER-Matrix werks- oder kundenseitig programmierbar. Zur Vereinfachung der kundenseitigen Programmierung wurden logische Felder entwickelt mit einer Programmiermöglichkeit nur für die UND-Matrix bei festgelegtem ODER-Feld. Dieser Typ des logischen Feldes wird auch als **PAL (Programmable Array Logic)** bezeichnet. (Die Bezeichnungsweise PAL, PLA, etc. ist nicht besonders systematisch.) Die Skizze zeigt ein Beispiel für ein solches logische Feld mit programmierbarer Eingangs-(UND-) Matrix.

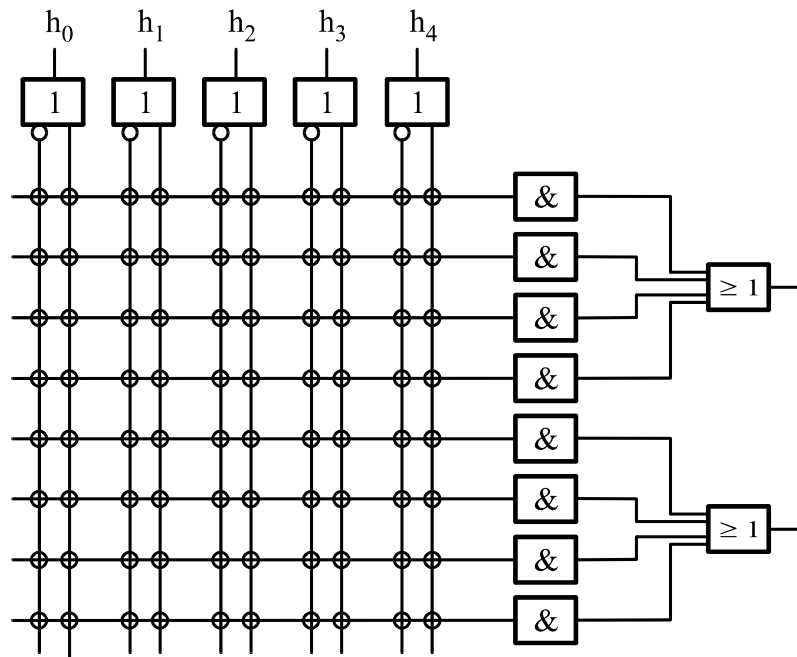


Bild 5.13: PAL-Blockschaltbild

Für den Einsatz als Schaltwerke sind PAL Schaltkreise häufig mit Ausgangs-D-Registern und entsprechender Rückführung ergänzt (Skizze)

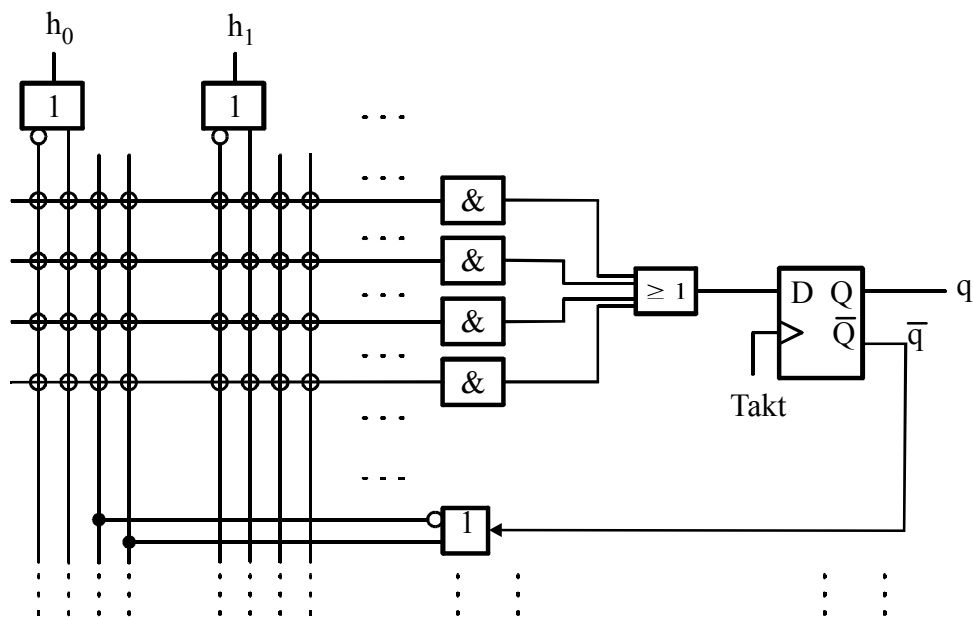


Bild 5.14: PAL-Schaltung mit D-Registern und Rückführung

5.3.1 Tabellen mit Hilfe von Festwertspeichern und logischen Feldern

An der Spitze der technologischen Entwicklung in der Großintegration stehen stets die verschiedenen Speicherschaltkreise. Derzeit sind z.B. bis zu 4GB-RAM Speicher (dynamisch) verfügbar. Diese Situation macht den Einsatz von Tabellenkomponenten ("Table-Look-Up") zunehmend interessant für die Abspeicherung aller möglichen Ergebnisse auch komplexer Algorithmen (z. B. Multiplikation) und die Adressierung dieser Tabellenspeicher durch die Variablen.

Als ein Beispiel sei eine einfache 2er Komplement-Multiplikation betrachtet. Es gilt für das Produkt P zweier Zahlen A und B:

$$\begin{aligned}
 P &= A \cdot B = (-b_N \cdot 2^N + \underbrace{\sum_{j=0}^{N-1} b_j \cdot 2^j}_{|B_j|}) \cdot (-a_N \cdot 2^N + \underbrace{\sum_{k=0}^{N-1} a_k \cdot 2^k}_{|A_k|}) \\
 &= -p_{2N} \cdot 2^{2N} + |P_i| \\
 P_i &= |A_k| \cdot |B_j| + a_N \cdot b_N \cdot 2^{2N} - a_N \cdot 2^N \cdot |B_j| \\
 &\quad + p_{2N} \cdot 2^{2N} - b_N \cdot 2^N \cdot |A_k| \\
 p_{2N} &= a_N \neq b_N
 \end{aligned}$$

Für ein Produkt mit einem festen Koeffizienten B z. B. sind dann die Adressen durch $\{a_0, a_1, \dots, a_N\}$ gegeben und die Speicherinhalte berechenbar (s. o.). Für Wortlängen für A und B mit (N+1) ergeben sich Produktwortlängen für P mit 2N+1 und es ist die zur Verfügung stehende Tabellenspeicherkapazität:

$$C_1 = 2^{N+1} \cdot (2N + 1)$$

Wählt man demgegenüber 2 Adressvariable, d.h. für ein Produktspeicher mit variablen Koeffizienten, so ergibt sich die Speicherkapazität zu

$$C_2 = 2^{2(N+1)} \cdot (2N + 1)$$

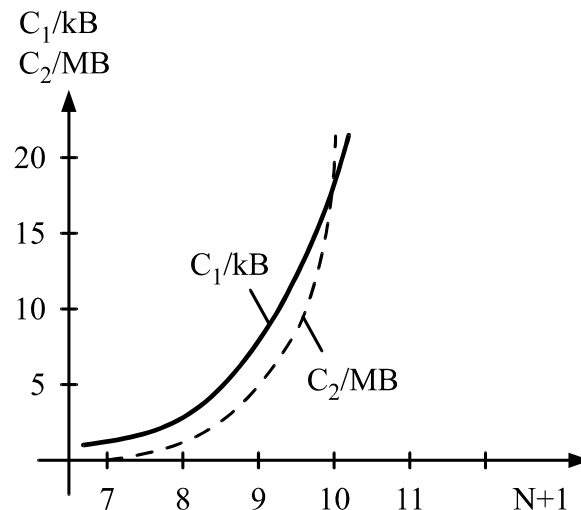


Bild 5.15: Erforderliche Kapazität von Tabellenspeichern für eine Zweierkomplement-Multiplikation

Speicherbedarf für Produkttabellenspeicher $(N+1) \times (N+1)$ Bit mit 2 bzw. 1 Eingangsvariablen (fester bzw. variabler Koeffizient).

Der exponentielle Anstieg des Speicherbedarfs ist ersichtlich. Für eine 8x8-Bit Produkttabelle ergibt sich z. B. für feste Koeffizienten etwa $C_1 \approx 4 \text{ kb}$, für variable Koeffizienten $C_2 \approx 1 \text{ Mb}$. Insbesondere im letzten Fall sind auch die heutigen Technologiemöglichkeiten bei höheren Wortlängen schnell erreicht.

An einem einfachen Tabellenbeispiel sei der Einsatz von Festwertspeichern noch dargestellt. Es sei eine Tabelle für eine Quadrierung von 2er Komplementzahlen gefordert:

$$Q = A \cdot A = |A_k| \cdot |A_k| + a_N 2^{2N} - (a_N \cdot 2^N \cdot |A_k|) \cdot 2$$

$$= |Q_1| \quad \text{da } q_{2N} = 0$$

Für $(N+1) = 3$ -stellige Zahlen A ergibt sich folgende Wertetabelle:

A	$Q = A^2$	A (Adresse)			Q (Ergebnis)				
		a_1	a_2	a_3	q_4	q_3	q_2	q_1	q_0
-4	16	1.	0	0					
-3	9	1.	0	1	0.	1	0	0	1
-2	4	1.	1	0	0.	0	1	0	0
-1	1	1.	1	1	0.	0	0	0	1
0	0	0.	0	0	0.	0	0	0	0
1	1	0.	0	1	0.	0	0	0	1
2	4	0.	1	0	0.	0	1	0	0
3	9	0.	1	1	0.	1	0	0	1

Offensichtlich kann $q_4 = q_1 = 0$ gesetzt werden, ebenso ist $q_0 = a_0$ direkt verwendbar. Das bedeutet mit den 3 Adreßvariablen a_2, a_1, a_0 kann direkt ein PROM bzw. PLA-Festwertspeicher 8x2 bzw. 4x2 Bit adressiert werden.

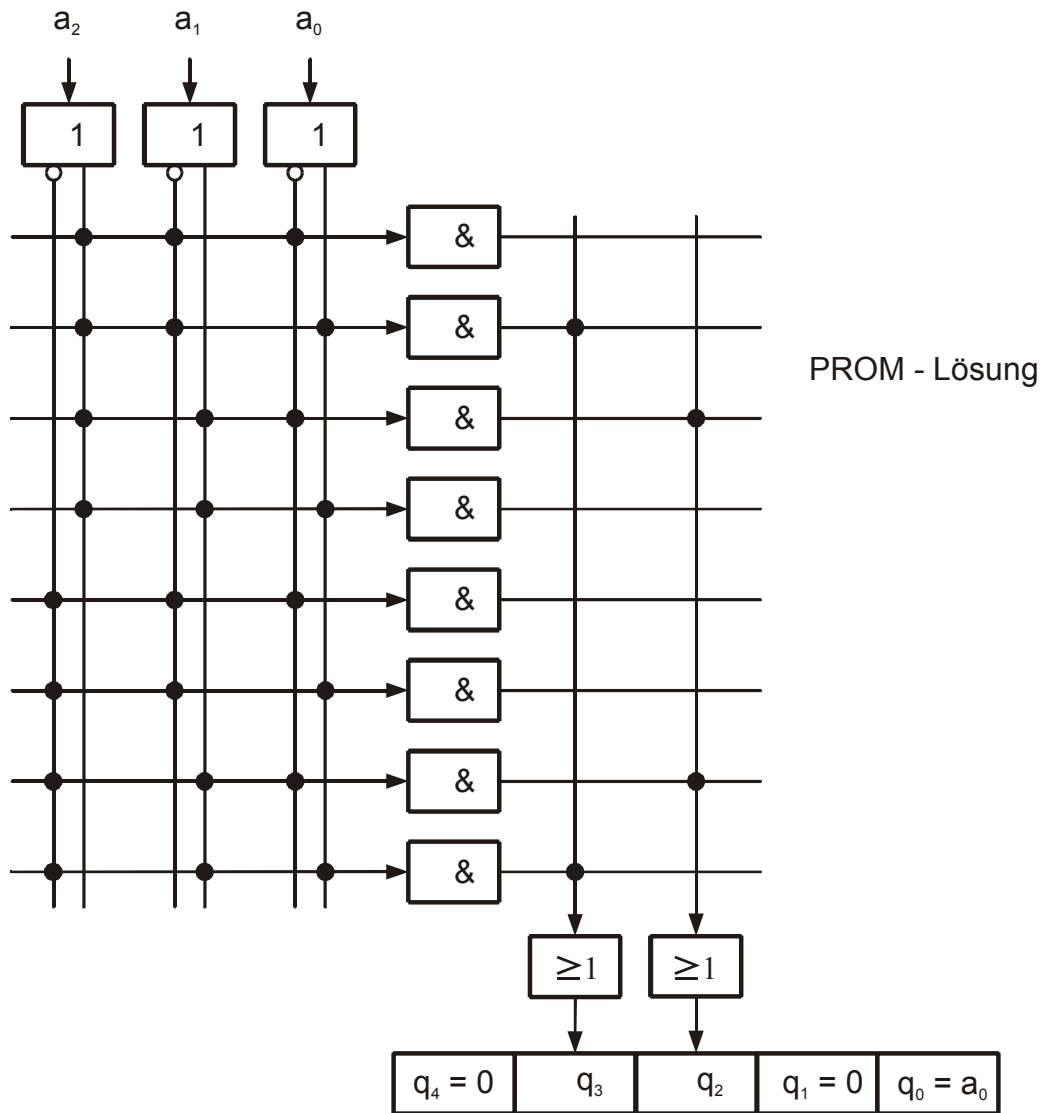


Bild 5.16: PROM-Schaltung als Quadratzahltable

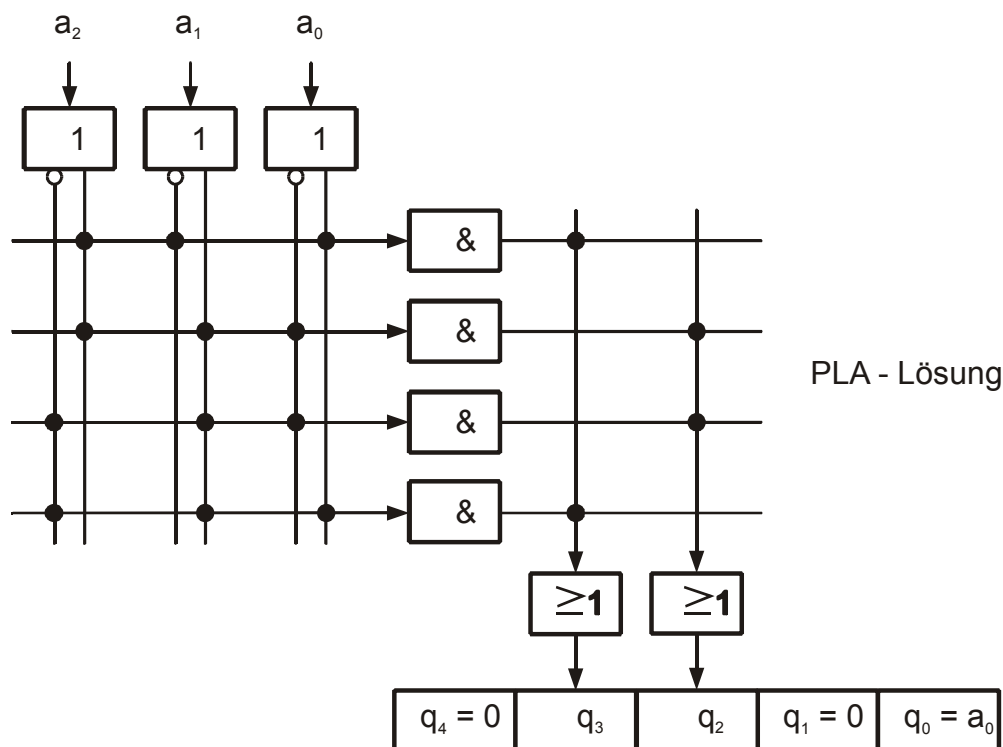


Bild 5.17: PLA-Schaltung als Quadratzahltable

Ersichtlich müssen bei der PROM-Lösung sämtliche möglichen Adresskombinationen berücksichtigt werden, während beim PLA in diesem Beispiel nur die Hälfte der möglichen Adresskombinationen realisiert werden muss.

5.4 Übersicht: Anwenderprogrammierbare Schaltungen und logische Felder

5.4.1 Einordnung bei den integrierten Schaltungen

Um die verschiedenen Implementierungsformen zu erläutern und einzuordnen, wird zuerst eine grobe Klassifikation integrierter Schaltungen in Form eines Baumdiagramms vorgenommen. Hierbei erfolgt eine Einteilung in verschiedene Gruppen und hierarchische Ebenen (Bild 5.18). Wie an diesem Baumdiagramm erkennbar ist, erfolgt zuerst eine Einteilung der integrierten Schaltungen in *Standard-ICs* und *anwendungsspezifische ICs*.

Die *Standard-ICs* lassen sich wiederum funktionell unterteilen in die *festverdrahteten ICs* (μ -Prozessoren, Standard-Systemelemente wie z.B. E/A-Bausteine etc. und Speicherbausteine) und die *maskenprogrammierbaren ICs* (wie z.B. Festwertspeicher, ROMs). All diese Bauelemente werden vom IC-Hersteller entworfen, sind universell einsetzbar und können in großen Stückzahlen vorgefertigt werden.

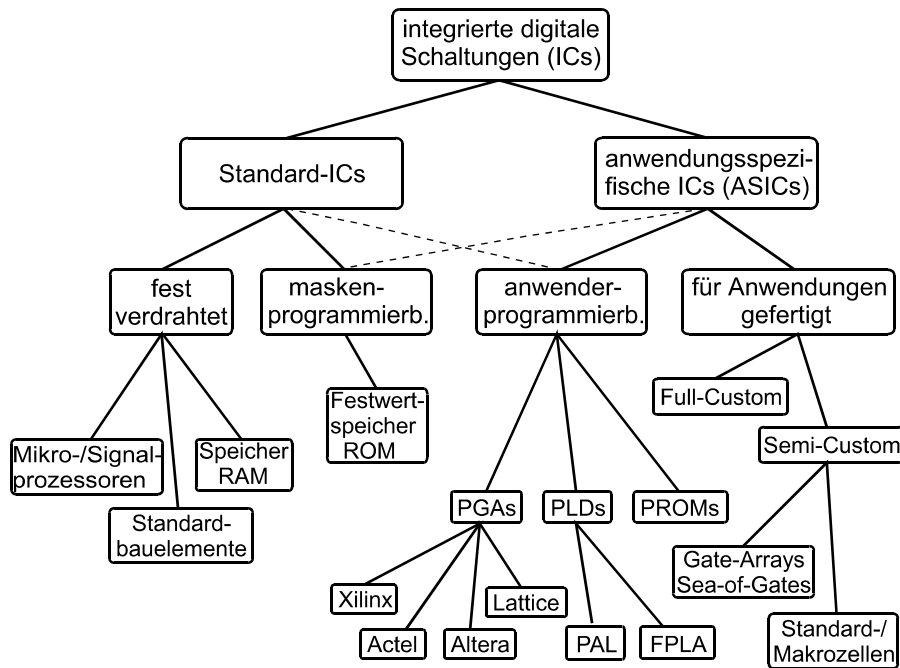


Bild 5.18: Klassifikation integrierter Schaltungen

Die *anwendungsspezifischen ICs* können weiter unterteilt werden in *anwenderprogrammierbare* Bauelemente und in *für Anwendungen gefertigte* Bauelemente. Bei den anwenderprogrammierbaren Bauelementen handelt es sich um universelle Elemente, deren Fertigung vom Hersteller schon völlig standardisiert vorgenommen wurde und die erst durch eine vom Anwender vorzunehmende Programmierung auf ihre endgültige Funktion festgelegt werden. Die für spezielle Anwendungen gefertigten ICs können noch weiter unterteilt werden in *Full-Custom-Entwürfe* und *Semi-Custom-Entwürfe*. Beim Full-Custom-Entwurf werden alle Entwurfsschritte entwurfsspezifisch parametrisiert und optimiert, und es gibt keinerlei Einschränkungen für den Entwurf. Bei Semi-Custom-Entwürfen wird auf bereits vorgefertigte Basisstrukturen zurückgegriffen, und diese werden dann nur anwendungsspezifisch miteinander verdrahtet (Standardzellen, Makrozellen, Gate-Arrays, Sea-of-Gates).

Da die Verwendung integrierter Schaltungen seit Jahren zunimmt, werden große Anstrengungen unternommen, um eine möglichst effiziente (flächensparende, verlustleistungsarme usw.) Realisierung eines ICs in einer möglichst kurzen Zeit zu erreichen. Das Bild 5.19 zeigt, daß es sich hierbei aber um zwei gegensätzliche Tendenzen handelt. Über verschiedenen Klassen von Fertigungsstrukturen, die im weiteren Verlauf des Kapitels diskutiert werden, sind der benötigte Flächenaufwand sowie die erforderliche Entwicklungszeit aufgetragen. Es ist zu erkennen, daß die Chipfläche für eine gegebene Anwendung nur durch eine Erhöhung der

Entwicklungszeit reduziert werden kann. Die Chipgröße sowie die aufgebrachte Entwicklungszeit bestimmen aber in entscheidendem Maße die Kosten für ein IC.

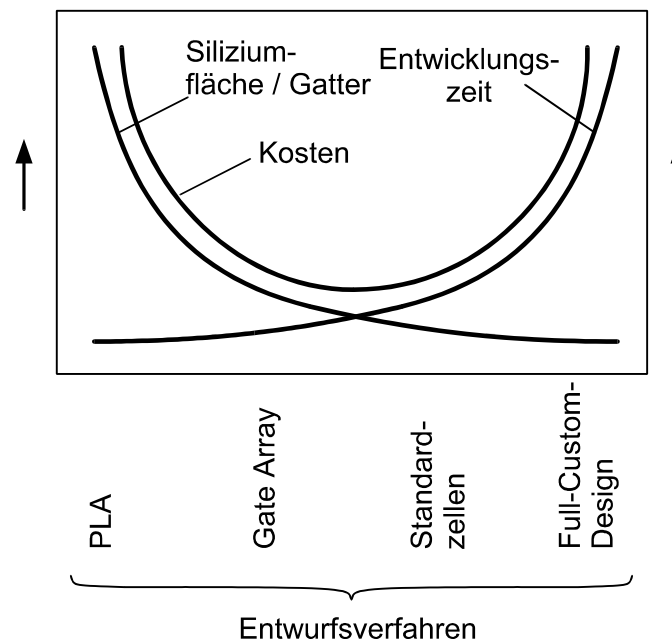


Bild 5.19: Entwurfs- und Fertigungsparameter bei integrierten Schaltungen

Für einen wirtschaftlichen Vergleich der unterschiedlichen Entwurfsstile sind die für einen ASIC entstehenden *Entwicklungs-* und *Stückkosten* maßgebend.

Die *Entwicklungskosten* eines ASIC setzen sich aus der erforderlichen Zeit zur Logik-Definition, Simulation und Verifikation sowie aus der Zeit zur Erstellung des Layouts¹ zusammen. Beim *Gate-Array-* und *Standardzellen-Entwurf* werden die Schaltungsfunktionen mit Hilfe von Hardware-Beschreibungssprachen (z.B. VHDL oder Verilog) definiert und durch Synthesewerkzeuge automatisch in Gatternetzlisten umgesetzt. Alle Makrozellen dieser Netzlisten werden vordefinierten Zellbibliotheken entnommen. Die Platzierung und Verdrahtung der Zellen erfolgt automatisch unter Zuhilfenahme von CAD²- und CAE³-Entwicklungssystemen. Auf diese Weise sind

¹Layout = Die Menge aller Dokumente, die als Vorlage für die Fertigung eines integrierten Schaltkreises dienen. Für jeden Fertigungsschritt wird dabei eine eigene Vorlage erstellt, die als Maske bezeichnet wird. Jede Maske beinhaltet eine Menge von nicht überlappenden geometrischen Strukturen. Die Beziehungen der geometrischen Strukturen auf den verschiedenen Masken zueinander bestimmen elektronische Bauelemente und deren Verbindung (siehe auch [MeadCon80]).

² CAD = **C**omputer **A**ided **D**esign

³ CAE = **C**omputer **A**ided **E**ngineering

Entwicklungszeiten und -kosten dieser beiden halbkundenspezifischen Entwürfe annähernd gleich.

Ein *Full-Custom-Design* hingegen wird aus einzelnen, optimierten Transistor-komplexen aufgebaut. Eine automatische Platzierung und Verdrahtung der Transistoren ist in vielen Fällen unmöglich. Daher können für Schaltungsentwürfe mit einem hohen Anteil an nichtregulärer Logik die Entwicklungszeiten (im ungünstigen Fall ein bis zwei Jahre) und daraus folgend auch die Entwicklungskosten weitaus größer sein als bei einem entsprechenden *Semi-Custom-Design*.

Diese Kostenfaktoren zur Entwicklung eines vollkundenspezifischen Entwurfs ändern sich jedoch, wenn ein reguläres Design entwickelt wird. In diesem Fall können auch vollkundenspezifische Strukturen zu einer *Makrozellen-Bibliothek* zusammengefaßt und die Zellen schnell und einfach auf einem Chip platziert und verdrahtet werden. Die Entwicklungszeiten derartiger Entwürfe sind nicht wesentlich länger als bei gleichartigen Gate-Array- oder Standardzellen-Entwürfen.

Aufgrund der vollständig optimierten Struktur können bei Full-Custom-Schaltungen die Chipflächen im Vergleich zum Semi-Custom-ASIC noch weiter reduziert werden. Die Silizium- und Prozeßkosten pro Chip sind folglich bei Gate-Arrays höher als bei den anderen ASIC-Entwürfen. Für die Fertigung großer Mengen ist es deshalb vorteilhaft, auf Full-Custom-Schaltungen zurückzugreifen.

5.5 FPLD: Field Programmable Logic Device – Übersicht

Die prinzipielle Architektur eines FPLDs zeigt Bild 5.20.

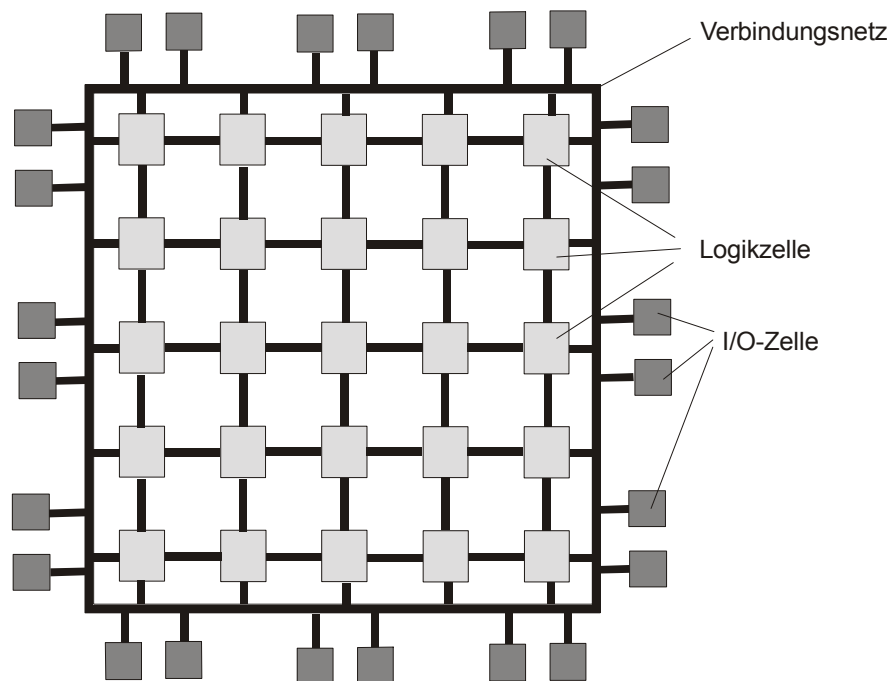


Bild 5.20: FPLD Blockschaltbild (Prinzip)

Es ist hierin eine Anzahl von Logikzellen enthalten, die zur Realisierung von Logikfunktionen verwendet werden. Die Logikzellen sind matrixförmig angeordnet. Verbindungen zwischen diesen und nach außen über die I/O-Blöcke sind mit Hilfe des Verbindungsnetzes möglich.

Auf der Basis dieser grundsätzlichen Struktur gibt es unterschiedliche Varianten hinsichtlich Programmierbarkeit, interner Logikzellen-Struktur und des Verbindungsnetzes. Logikzellen können aus einem Transistor oder einem komplexen Mikroprozessor bestehen. Die Logikzelle kann kombinatorische oder sequentielle Funktionen unterschiedlicher Komplexität übernehmen. Kommerzielle FPLDs besitzen Logikzellen basierend auf

- Transistorpaaren
- Basisgattern, z.B. NAND oder EXOR (2 Eingänge)
- Multiplexer
- Tabellen (Look-Up-Table: LUT)
- große AND-OR-Felder

Drei wesentliche Programmierkonzepte sind für FPLDs verbreitet

- EPROM –programmierte Verbindungsnetze
- Antifuse-Verbindungen
- Statische RAMs (Random Access Memory), die beim Einschalten geladen werden müssen.

Dabei sind im wesentlichen drei Hauptgruppen von FPLDs in entsprechenden Kombinationen von Architekturen und Programmierkonzepten gebräuchlich

- EPROM / Complex Programmable Logic Devices (EPLDs, CPLDs)
- Antifuse FPGAs
- SRAM basierte anwenderprogrammierbare Logikfelder: SRAM FPGAs (FPGA: Field-Programmable Gate Array)

5.6 EPROM / Complex Programmable Logic Devices (EPLDs, CPLDs)

Diese Bausteine basieren auf Weiterentwicklungen von PAL-Architekturen, die aus einer programmierbaren UND und einer festen ODER-Matrix bestehen (Bild 5.21).

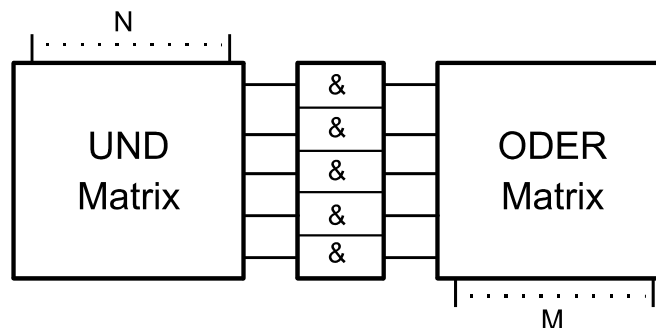


Bild 5.21: UND/ODER-Matrix als PLD-Basis

Diese sind in Funktionsblöcken realisiert und in diesen sind die ODER-Zweige mit flexiblen Makrozellen ergänzt. Programmiert werden über ein Verbindungs-EPROM oder EEPROM das Verbindungsnetz und die Makrozellen (Bild 5.22).

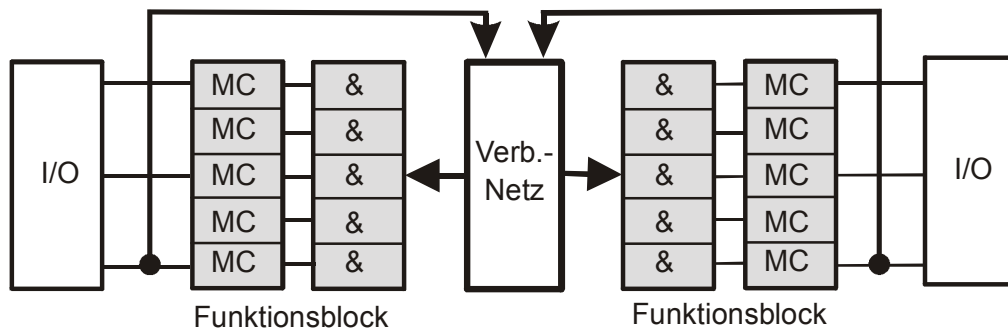


Bild 5.22: Typische CPLD-Grundstruktur (CPLD, EPLD)

Das UND-Feld enthält eine Anzahl von Produkttermen. Die Programmierung gestattet über EPROM-Transistoren die Auswahl der Variablen in den Produkttermen. Die programmierbaren Makrozellen enthalten neben dem ODER-Gatter auch Register und I/O-Blöcke. Ebenso können Multiplexer und ihre EPROM-Zellen enthalten sein, um Daten aus unterschiedlichen Quellen bzw. Stufen zu schalten vgl. die vereinfachte Makrozelle (Bild 5.23).

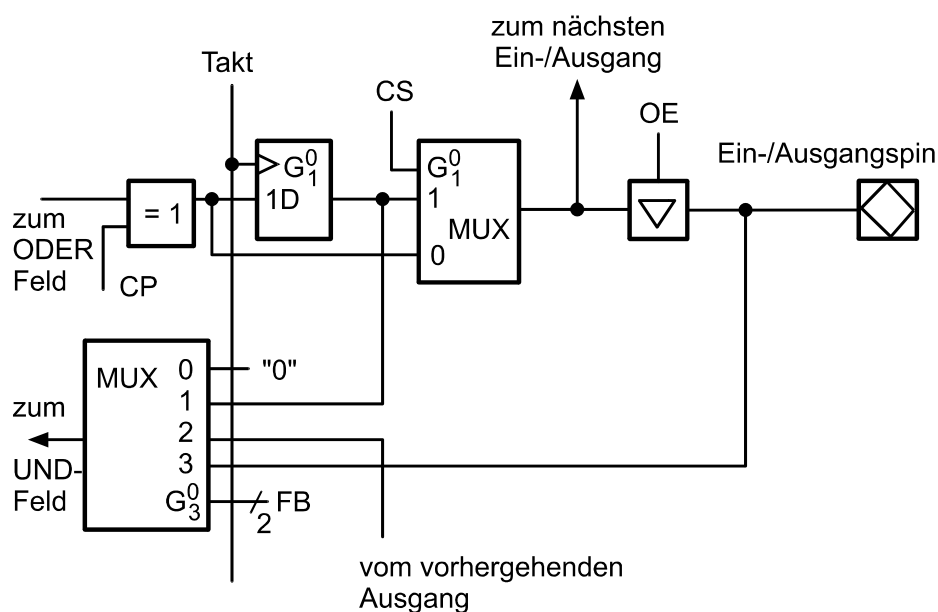


Bild 5.23: Vereinfachte Makrozelle (programm. Ausgangsmakro)

Diese größere Flexibilität läßt sich an den nun zusätzlich verfügbaren Funktionen erläutern:

- Das EXOR-Gatter am Ausgang des ODER-Feldes erlaubt es, das Ausgangssignal durchzulassen (CP=0) oder zu komplementieren (CP=1).
- Der durch CS gesteuerte Multiplexer erlaubt es, das Ausgangssignal direkt durchzulassen (CS=0) oder erst im D-Flip-Flop zwischenspeichern (CS=1).

- Der durch OE gesteuerte Treiber erlaubt es, den Ausgang zu deaktivieren und ihn als Eingang zu benutzen.
- Durch die zwei FB-Bits kann ein weiterer MUX gesteuert werden und entweder das Eingangs- oder ein Ausgangssignal wieder in das UND-Feld zurückgekoppelt werden (wichtig ist hierbei, daß die maximale Verzögerungszeit berücksichtigt wird !).

Die hier exemplarisch gezeigten Möglichkeiten zur Programmierung einfacher Ausgangsmakros erhöhen die Flexibilität programmierbarer logischer Felder erheblich. Die einzelnen herstellerepezifischen Ausgangsmakros unterscheiden sich dabei in ihren Programmiermöglichkeiten.

Eine weitere Flexibilisierung bei PAL-Strukturen ist erzielbar, wenn man zu einer flexiblen Produktterm-Allokation übergeht (Bild 5.24). Bei PALs kann jeder Produktterm nur einmal verwendet werden (d.h. der Ausgang jedes UND-Gatters wird nur an ein ODER-Gatter geführt). Die Maximalzahl der Produktterme pro Ausgabe (d.h. pro ODER-Gatter) ist somit typenspezifisch vorgegeben und fest. Ersetzt man die ODER-Gatter nun durch relativ einfache programmierbare Strukturen, wie das Bild 5.24 zeigt, so erreicht man eine wesentliche Flexibilisierung.

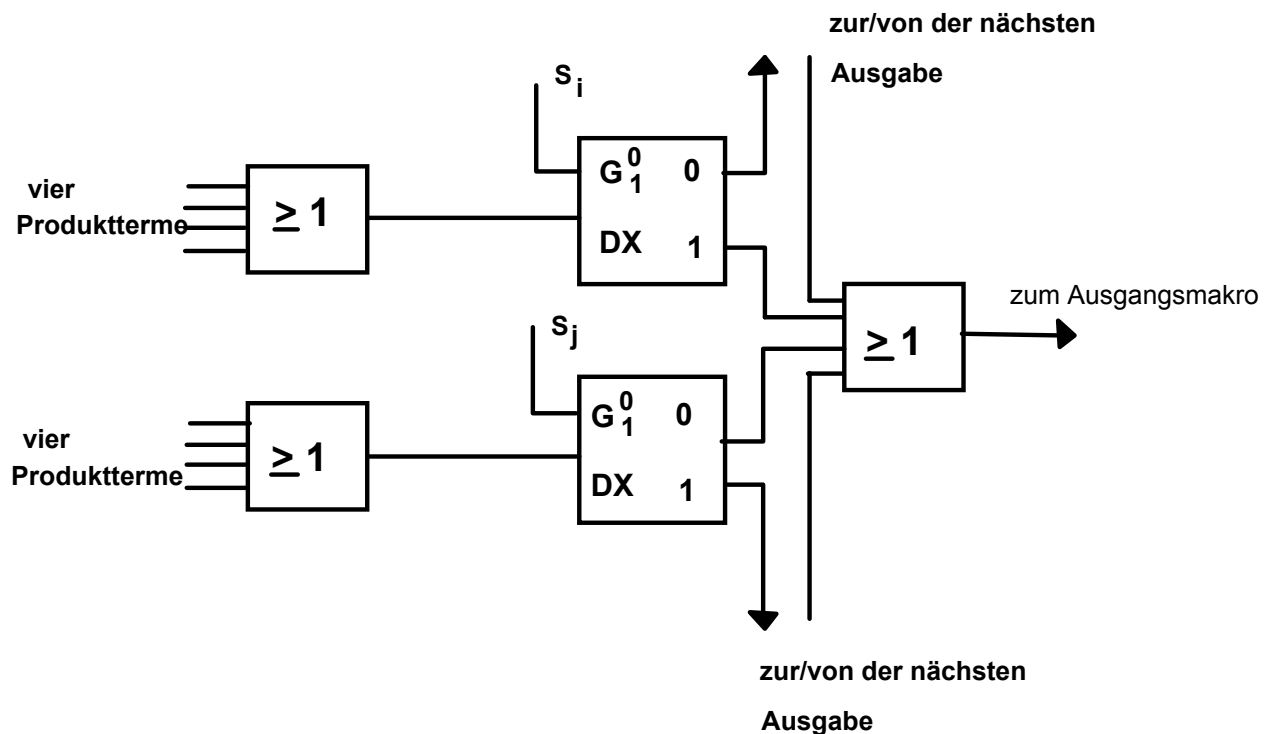


Bild 5.24: Flexible Produktterm-Allokation

In diesem Beispiel können durch Programmieren der Demultiplexereingänge S_i und S_j pro Ausgang bis zu 16 Produktterme verwendet werden.

5.6.1 CPLD-Bausteine, Beispiel: Altera 7000 EPLD

Die Grundstruktur des Logikblockes mit ODER-Gatter und programmierbarer Negation (EXOR) zeigt die folgende Skizze.

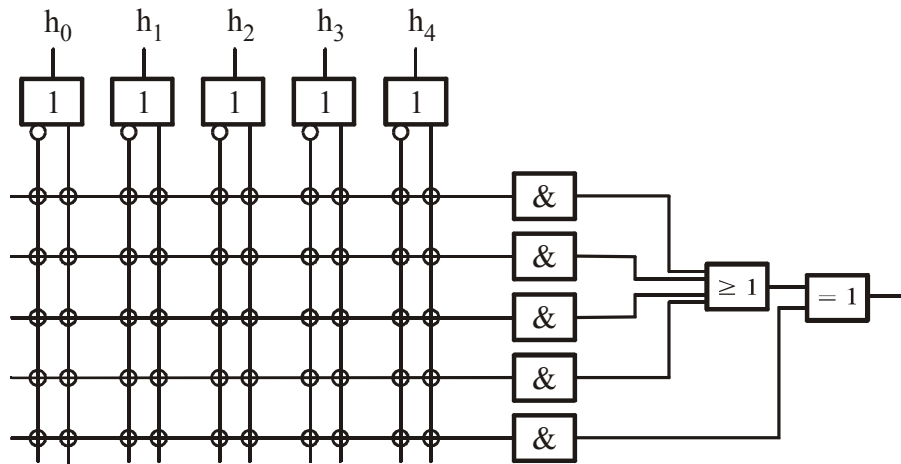
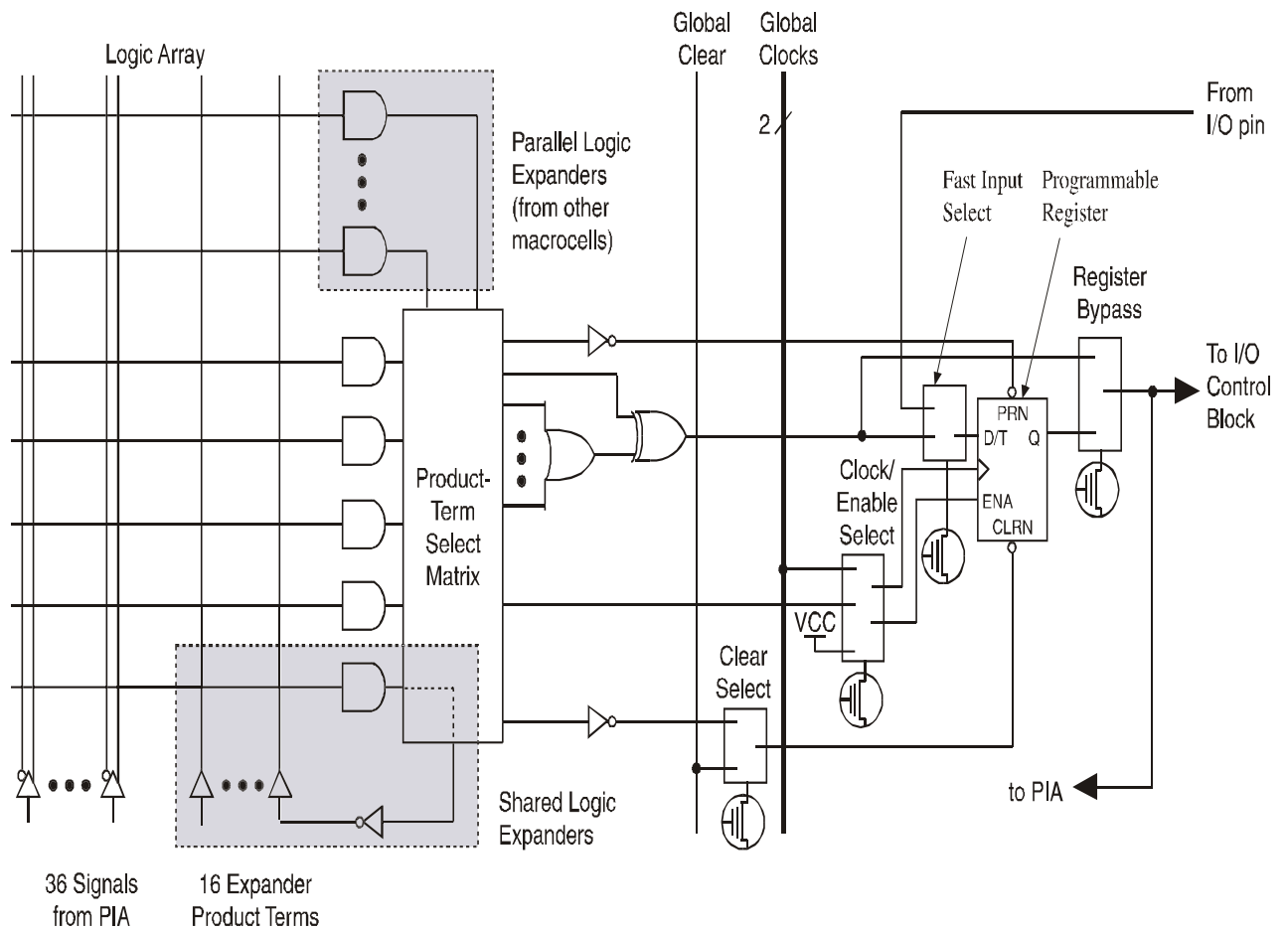


Bild 5.25: Logikblock mit programmierbarer Negation

Die komplette Makrozell-Architektur zeigt das Bild 5.26 aus dem Datenblatt der elektrisch programmierbaren (EPLD) MAX 7000er Serie.

EPM7032, EPM7064 & EPM7096 Device Macrocell

**Bild 5.26:** Makrozelle des Altera 7000 EPLD

Hinweis: Hier ist die amerikanische Logik-Norm verwendet, das eingekreiste Floating-Gate-Transistor-Symbol weist auf die jeweilige EPROM-Programmierbarkeit für die Signalmultiplexer hin (Select-Bausteine). Die Gesamtarchitektur zeigt das Bild 5.27.

MAX 7000E & MAX 7000S Device Block Diagram

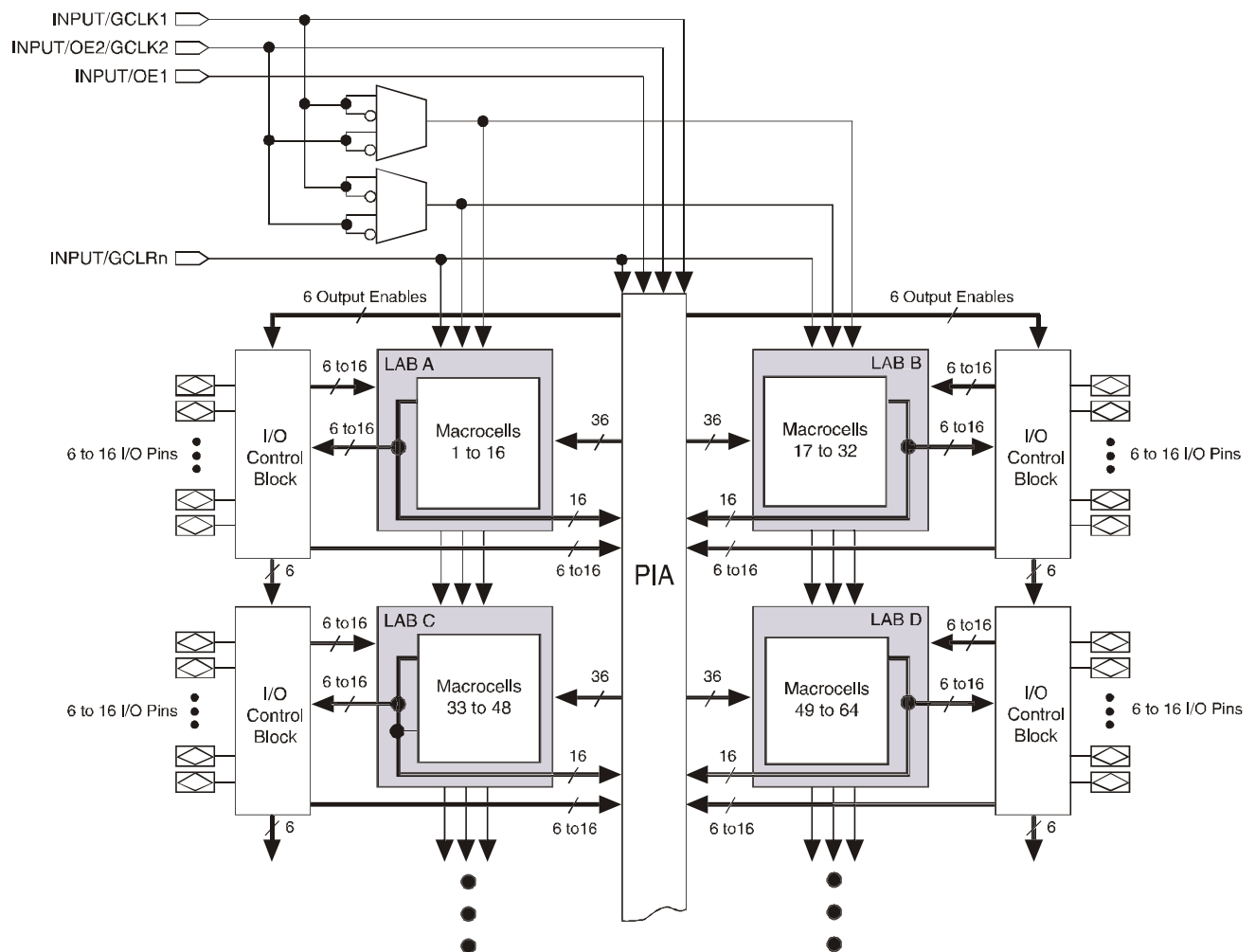


Bild 5.27: Blockstruktur von Altera 7000 EPLD

Es sind Funktionsblöcke mit jeweils 16 Makrozellen zusammengefügt, die über ein Programmable Interconnect Array (PIA) verbunden sind. Bausteine der MAX 7000-Reihe sind verfügbar mit 32, 64, 96, 128, 160, 192, 256 Makrozellen (entsprechend 2, 4, 6, 8, 10, 12, 16 Funktionsblöcken).

5.7 Antifuse-FPGA, Beispiel: ACTEL-Bausteine

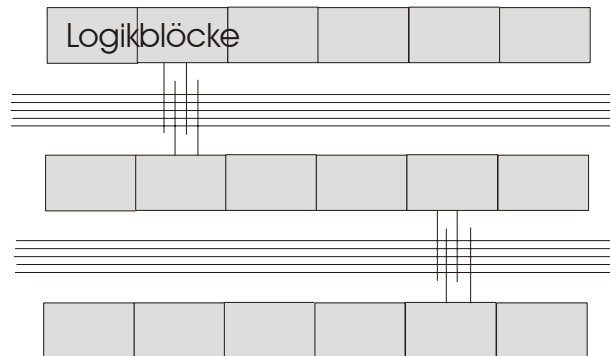


Bild 5.28: Antifuse FPGA

„Antifuses“ sind Verbindungen, die durch sehr hohe Spannung eine permanente niederohmige Verbindung zwischen zwei Verbindungspunkten erzeugen. Architekturen sind dann zweckmäßig reihenorientiert (s. Bild 5.28).

Bei den Actel-Bausteinen handelt es sich um sogenannte Multiplexer-basierte Bausteine. Im Unterschied zu den Altera oder Xilinx-PGAs können Actel-PGAs nur einmal programmiert und nicht wieder gelöscht werden. Andererseits geht ihr Inhalt bei Abschalten der Versorgungsspannung auch nicht verloren. Eine Programmierspannung zerstört ein zwei Leitungen voneinander trennendes Dielektrikum (Antifuses). Es handelt sich also um das umgekehrte Prinzip wie bei EPROMs.

Die Grundstruktur dieser Bausteine ähnelt ebenfalls der der Gate-Arrays. Es befinden sich auf den Mastern streifenförmig angeordnete Funktionszellen und dazwischenliegende Verdrahtungskanäle. Am Chiprand gibt es wie bei Xilinx-Bausteinen konfigurierbare I/O-Zellen. Bei den Funktionszellen im Kernbereich wechseln kombinatorische und sequentielle Zellen sich ab. Die sequentiellen Zellen enthalten hierbei Speicherzellen. Das Bild 5.29 verdeutlicht dies.

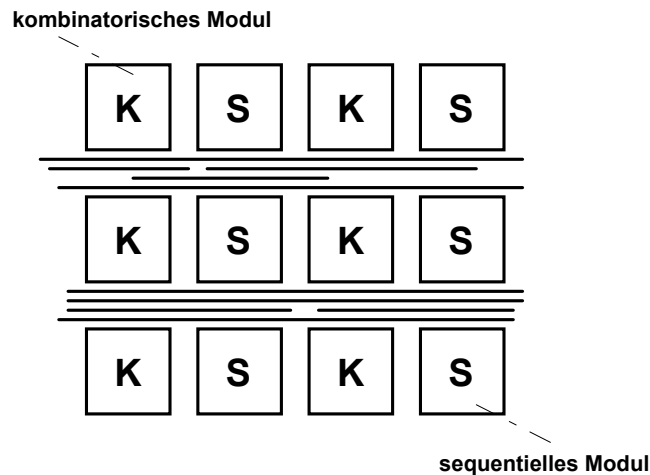


Bild 5.29: Kombinatorische/sequentielle Module bei ACTEL Antifuse FPGA

Im Bild 5.30 werden die Grundstrukturen eines kombinatorischen (links) und eines sequentiellen (rechts) Zellmoduls gezeigt.

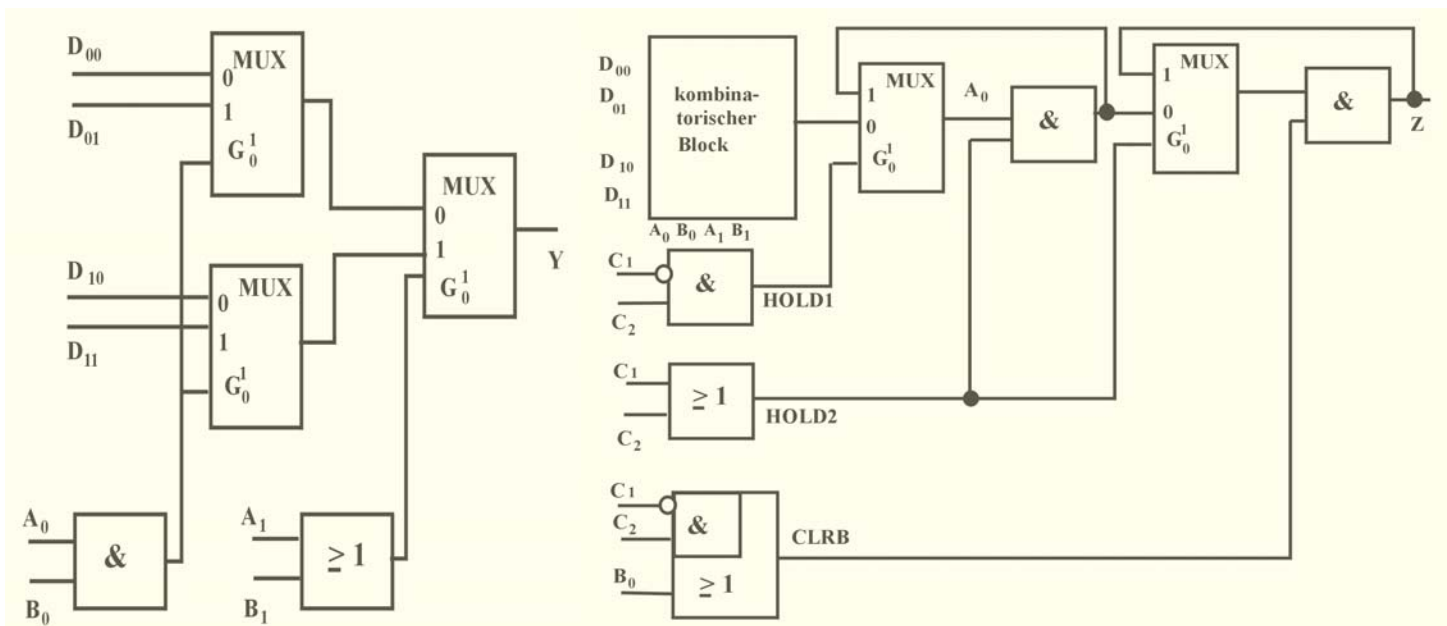


Bild 5.30: Grundstrukturen kombinatorisches (links) und sequentielles (rechts) Zellmodul

5.8 SRAM-basierte Bausteine, Beispiel: Xilinx-Bausteine

Im Folgenden soll beispielhaft für die SRAM-basierten Bausteine auf die Grundstruktur von Xilinx-FPGAs eingegangen werden. Die speicherbasierten Xilinx-Bausteine besitzen eine Topologie, wie sie im Bild 5.31 dargestellt ist.

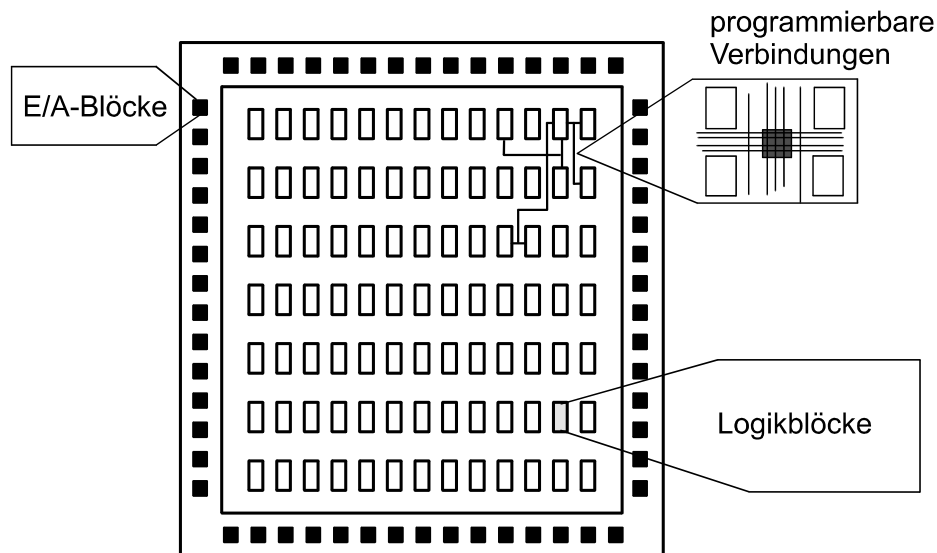


Bild 5.31: Grundstruktur Xilinx 3000/4000 FPGA

Diese Topologie ähnelt sehr der Grundstruktur von Gate-Arrays. Im Kern des ICs finden sich Inseln von Transistorgruppen. Bei den Xilinx-Bausteinen handelt es sich um *SRAM-basierte Bausteine*, da die Funktionszellen im Kernbereich jeweils eine zweistufige SRAM-Speicherstruktur mit wenigen Eingängen realisieren. Der Inhalt der SRAMs muß vor Beginn der Benutzung jeweils neu einprogrammiert werden. Mit diesen Speicherzellen kann so eine beliebige logische Funktion (mit fester Eingangszahl) realisiert werden. Da sich zwischen den Zellen programmierbare Verbindungsmatrizen befinden, ist eine flexible Verbindung der Zellen untereinander möglich. Den Randbereich des Chips nehmen programmierbare Ein-/Ausgabezellen ein, die über die Verbindungsmatrix an die Funktionszellen angeschlossen werden können.

Das Bild 5.32 zeigt als Beispiel eine vereinfachte Funktionszelle der XC 3000 Serie.

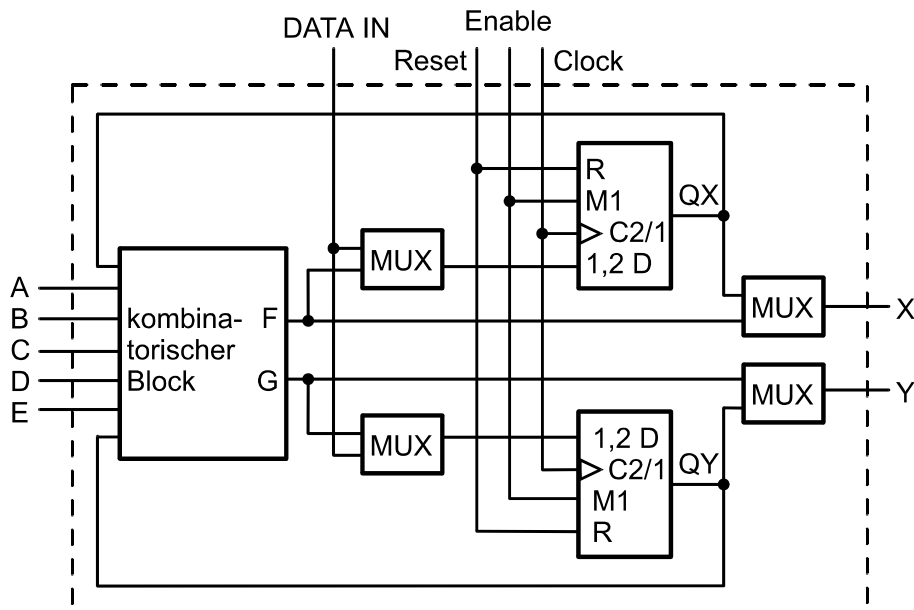


Bild 5.32: Funktionszelle beim XC 3000 FPGA

Diese Funktionszelle besitzt neun Eingänge und zwei Ausgänge. Mittels eines 32 Bit SRAMs (der kombinatorische Block im Bild 5.32) können zwei Funktionen (F und G) gebildet werden. Es ergeben sich damit Möglichkeiten, entweder eine Funktion zu bilden, die von fünf Eingängen abhängt ($F=G$) oder zwei Funktionen, die von vier Variablen abhängen. Im rechten Teil der Zelle erkennt man Multiplexer und D-Flip-Flops, die jeweils wieder ein direktes Durchschalten bzw. Zwischenspeichern oder Rücksetzen der Ergebnisse erlauben.

5.9 Bemerkungen zum Entwurf programmierbarer Bausteine

In diesem Abschnitt sollen kurz einige unterschiedliche Aspekte beim Entwurf einer Schaltung (beim sogenannten *Design Flow*) diskutiert werden. Dabei werden hier PGAs, PLDs und Gate-Arrays berücksichtigt. Beginnt man diese Betrachtung auf der Ebene der Netzlisten, so erhält man die Entwurfsstruktur entsprechend Bild 5.33.

Programmierbare Bausteine (also PGAs und PLDs) erlauben keine Realisierung von beliebigen Gattern, weshalb die Netzliste zunächst an den verwendeten Baustein angepaßt werden muß (also bei PLDs z.B. eine zweistufige und bei PGAs eine mehrstufige Realisierung einer logischen Funktion). Ein weiterer Unterschied ist, daß bei einem zweistufigen logischen Feld eine Platzierung und Verdrahtung der einzelnen Zellen entfällt. Dies muß bei PGAs und Gate-Arrays durchgeführt werden. Entscheidend ist auch, daß bei PLDs keine weitere Zeitanalyse aufgrund ihrer starren regelmäßigen Struktur durchgeführt werden muß. Der letzte Schritt beim

Entwurf unterscheidet diese drei Familien, indem bei PGAs und PLDs Programmierdaten und bei Gate-Arrays Fertigungsdaten für die letzten Metallisierungsebenen erstellt werden.

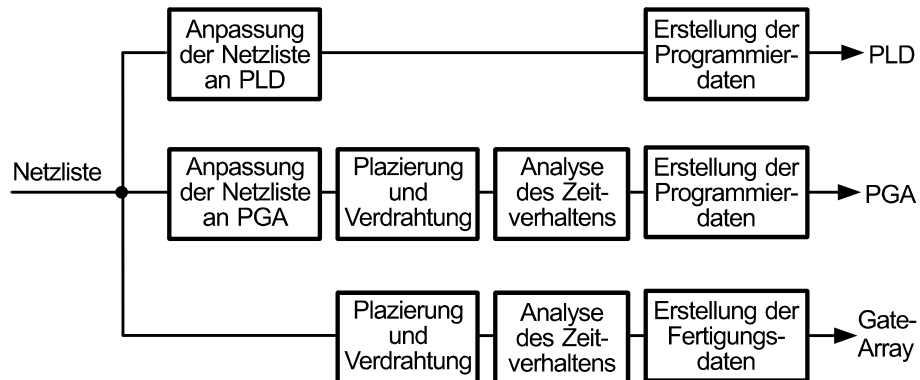


Bild 5.33: Entwurfsabläufe bei PLD, PGA, GA

5.9.1 Entwurfswerkzeuge

Entwicklungswerkzeuge stellen einen weiteren wichtigen Aspekt von FPGAs dar. Es muß berücksichtigt werden, daß eine FPGA-Architektur nur dann effizient ausgenutzt werden kann, wenn die zugehörige Entwicklungsumgebung in der Lage ist, einen Entwurf genau auf diese spezifische Architektur abzubilden. Die Entwicklungswerkzeuge für FPGAs umfassen dabei die Software, die es dem Designer erlaubt, die Logik zu definieren, sie zu verifizieren und sie in das FPGA abzubilden. Diese Werkzeuge sind unterteilt in sogenannte Front-End-Werkzeuge und Back-End-Werkzeuge. Eine grobe Partitionierung in diese Gruppen zeigt das Bild 5.34.

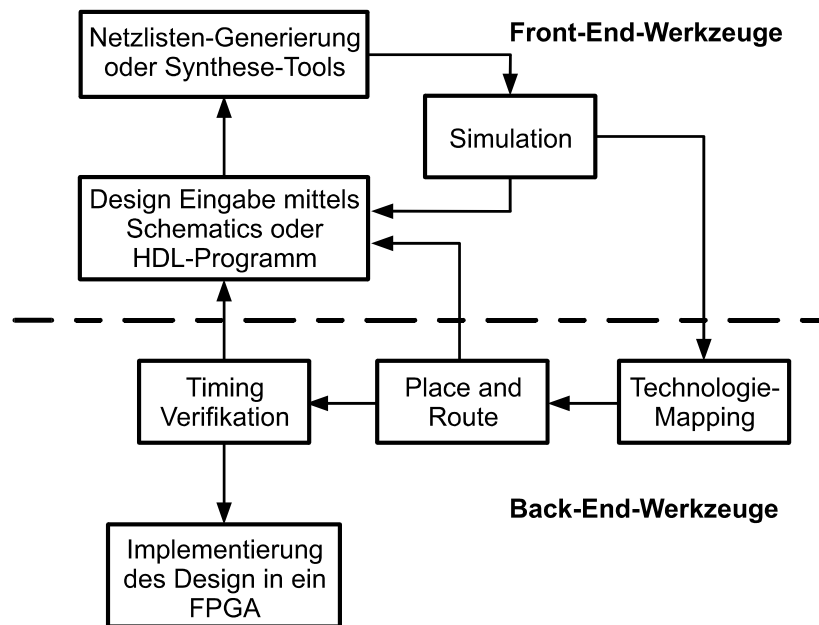


Bild 5.34: Entwurfs-Werkzeuge

Front-End-Werkzeuge erlauben die Eingabe eines Entwurfes, die erste Verifikation und die Generierung einer Netzliste. Die Eingabe eines Entwurfes erfolgt dabei in Form von sogenannten *Schematics* oder in Form einer Hardware-Beschreibungssprache (*HDL = Hardware Description Language*). Bei einer Eingabe mittels *Schematics* muß der Designer einen Entwurf auf einer Grafikoberfläche mittels Gattern und Logik-Elementen aus Bibliotheken mit primitiven Funktionen und Makros aufbauen. Bei einer Eingabe durch eine Hardware-Beschreibungssprache wird eine Programmiersprache verwendet, mit der die Funktionalität und die Eigenschaften primitiver Komponenten und Makros beschrieben werden können. Prominente Vertreter dieser Hardware-Beschreibungssprachen sind die Sprachen VHDL (Very High Speed IC Hardware Description Language) und Verilog. Diese Hardware-Beschreibungssprachen gewinnen zunehmend an Bedeutung, da ein Designer, der an den Umgang mit diesen Sprachen gewöhnt ist, für die Eingabe eines komplexen Designs mit Hilfe von HDLs weit weniger Zeit benötigt, als bei Eingabe über *Schematics*.

Eine Simulation des eingegebenen Designs findet an zwei Stellen im Ablauf des Entwurfes (dem sogenannten *Design Flow*) statt. Eine *Logiksimulation* nach der Generierung der Netzliste verifiziert die Funktionalität. An dieser Stelle ist der Entwurf noch nicht auf einen spezifischen Baustein abgebildet worden, weshalb nur geschätzte Verzögerungszeiten oder Einheitsverzögerungszeiten verwendet werden. Ein weiterer Punkt im Design Flow, an dem eine Simulation stattfindet, ist nach dem Plazieren und Routen des Designs. An dieser Stelle findet eine funktionelle Simulation des Entwurfes statt, bei der die exakten, dem abgebildeten Entwurf

entnommenen (im engl. *back annotated*), Verzögerungszeiten berücksichtigt werden.

Die im Bild 5.34 dargestellten Funktionalitäten eines *Back-End-Werkzeuges* enthalten die Abbildung (engl. *Mapping*) eines Designs auf einen spezifischen FPGA-Baustein, die Verifikation des Timings und die Konfiguration eines Programms zum Download in diesen Baustein beim Einschalten. Die Abbildungswerkzeuge konvertieren dabei die Originalelemente einer Netzliste in die verfügbaren logischen Elemente eines spezifischen FPGAs. Zugehörige Platzierungs- und Verdrahtungswerkzeuge (engl. *Place and Route Tools*) verbinden die Logikzellen untereinander und mit den Ein- und Ausgabe-Zellen. Design-Anforderungen wie z.B. die erforderliche Geschwindigkeit oder die maximale Verzögerung werden in diesem Schritt realisiert. Die Erfüllung dieser Design-Anforderungen hängt sowohl von der Effizienz der *Place and Route* Werkzeuge als auch von der Architektur des verwendeten Bausteins ab. Als letzten Schritt erzeugt das Back-End-Werkzeug einen Konfigurations-Bit-Stream, um die logische Funktion in den Baustein laden zu können. Dieser Schritt im Design Flow wird auch als *Bit-Stream-Generierung* bezeichnet. Back-End-Werkzeuge sind spezifisch für jeden FPGA-Baustein und werden daher von den einzelnen FPGA-Herstellern angeboten.