5 Komplexe Schaltnetze

5.1 Codierer

Funktion

Ein Codierer entspricht einem Schaltnetz, welches eine Menge von Eingangswerten in eine Menge von Ausgangswerten übersetzt (**Codeumsetzung**).

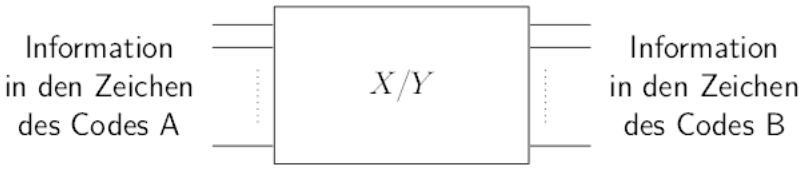


Abbildung 5.1: Schaltzeichen eines Codierers

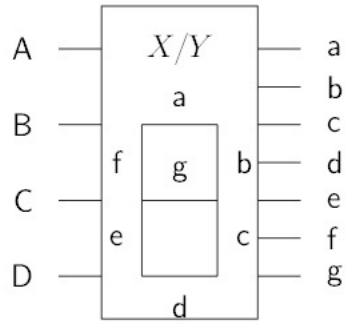
Das Schaltzeichen deutet an, dass ein Codierer ein Schaltnetz ist, das eine Vektorfunktion realisiert ($\overrightarrow{y} = \mathsf{f}(\overrightarrow{x})$). Die Anzahl der Ein- und Ausgänge eines Codeumsetzers ist abhängig von der Wortlänge des Binärcodes, in dem die Information dargestellt ist.

Beispiel 66. *Schaltnetzentwurf für die 8421-BCD zu 7-Segment-Umsetzung*

Aufgabe:
Dezimalziffern werden häufig in einem ***BCD-Code*** *(Binary Coded Decimal) dargestellt und in 7-Segmenteinheiten eingesetzt. Dies kann durch einen Codeumsetzer realisiert werden.*

Annahme:
Die Dezimalziffern liegen im 8421-BCD-Code vor.

Codeumsetzung:
1. Die Zuordnung (Codierung) der Dezimalziffern vom 8421-BCD-Code zu den Segmenten der 7- Segment-Anzeige wird in einer Wertetabelle festgelegt.



(a) Blockschaltbild

Dezimal-ziffer	8421-BCD-Code D	C	B	A	7-Segment-Code a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
10	1	0	1	0							
11	1	0	1	1							
12	1	1	0	0							
13	1	1	0	1							
14	1	1	1	0							
15	1	1	1	1							

(b) Wertetabelle

Abbildung 5.2: 8421-BCD-Code zu 7-Segment Codierer

2. Für die Ausgangsvariablen $\mathsf{a}, \mathsf{b}, ..., \mathsf{g}$ *werden die Funktionsgleichungen aus der Wertetabelle in der DNF hergeleitet. Nimmt man an, dass die Pseudotetraden nicht vorkommen (Zustand 10 bis 15), können diese bei der Vereinfachung im KV-Diagramm als don't care-Terme benutzt werden.*

3. Nach der Minimierung lauten die Funktionsgleichungen für die Ausgangsvariablen $\mathsf{a}, \mathsf{b}, ..., \mathsf{g}$ *in DNF:*

$$\mathsf{a} = \mathsf{D} + \overline{\mathsf{A}}\ \overline{\mathsf{C}} + \mathsf{A}\ \mathsf{C} + \mathsf{A}\ \mathsf{B}$$
$$\mathsf{b} = \overline{\mathsf{C}} + \mathsf{A}\ \mathsf{B} + \overline{\mathsf{A}}\ \overline{\mathsf{B}}$$
$$\mathsf{c} = \mathsf{A} + \overline{\mathsf{B}} + \mathsf{C}$$
$$\mathsf{d} = \overline{\mathsf{A}}\ \mathsf{B} + \overline{\mathsf{A}}\ \overline{\mathsf{C}} + \mathsf{B}\ \overline{\mathsf{C}} + \mathsf{A}\ \overline{\mathsf{B}}\ \mathsf{C}$$
$$\mathsf{e} = \overline{\mathsf{A}}\ \mathsf{B} + \overline{\mathsf{A}}\ \overline{\mathsf{C}}$$
$$\mathsf{f} = \mathsf{D} + \overline{\mathsf{A}}\ \overline{\mathsf{B}} + \overline{\mathsf{A}}\ \mathsf{C} + \overline{\mathsf{B}}\ \mathsf{C}$$
$$\mathsf{g} = \mathsf{D} + \overline{\mathsf{A}}\ \mathsf{B} + \overline{\mathsf{B}}\ \mathsf{C} + \mathsf{B}\ \overline{\mathsf{C}}$$

5.2 Decodierer

Funktion

Decodierer sind Codierer mit mehreren Ein- und Ausgängen, bei denen für jede Kombination von Eingangssignalen immer **nur je ein Ausgang** ein Signal abgibt. Am Ausgang eines Decodierers liegt die Information in den Zeichen eines 1-aus-n-Codes vor.

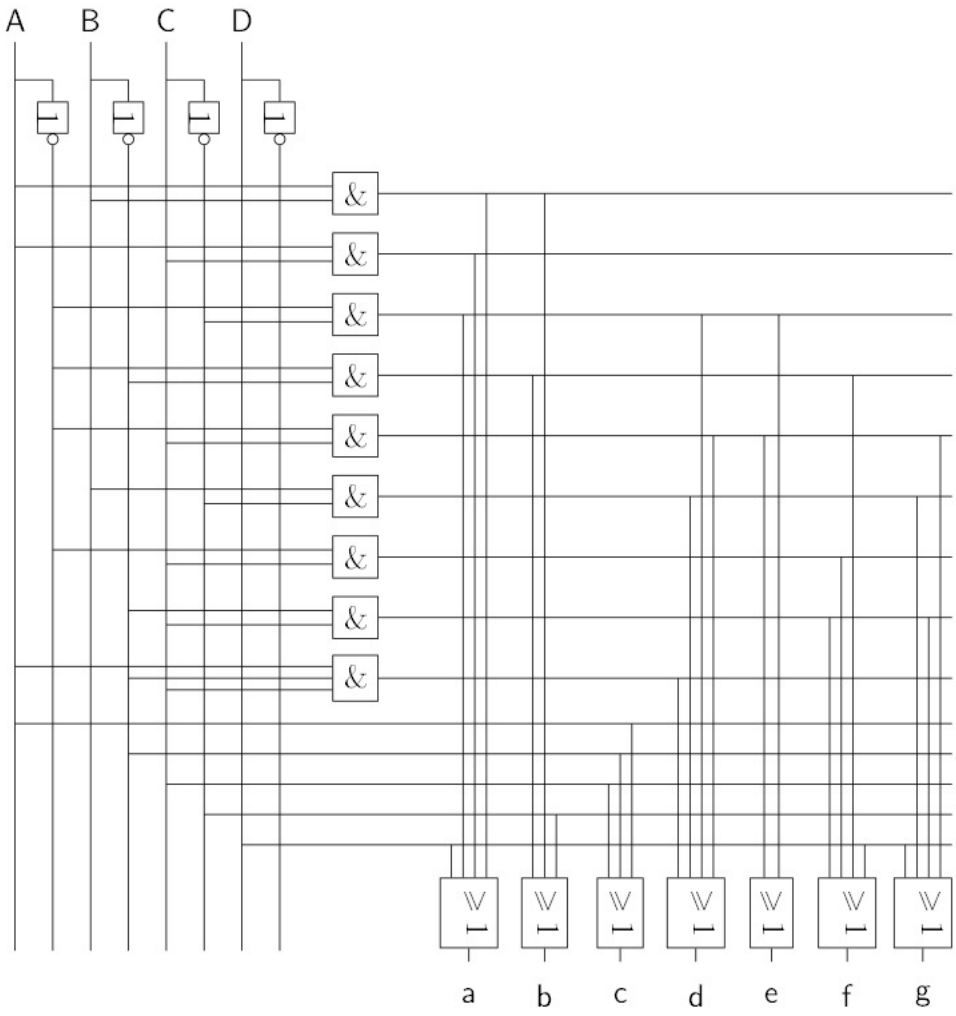


Abbildung 5.3: Schaltnetz für einen 8421-Code in 7-Segment Codierer

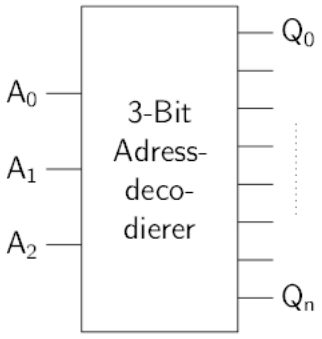
Definition 20. *1-aus-n-Code*
Ein 1-aus-n-Code ist dadurch gekennzeichnet, dass die Anzahl der Binärstellen gleich der Anzahl der darzustellenden Zeichen ist. D.h. führt eine Bitstelle des Codewortes ein 1- (bzw. 0-) Signal, dann führen alle anderen Stellen ein 0- (bzw.1-) Signal.

Beispiel 67. *3-Bit Adressdecodierer*

Über die Adresseingänge wird ein Ausgang des Decodierers angewählt, der dann 1-Signal führt (siehe Wertetabelle). Hat ein Adressdecodierer n*-Eingänge, dann können* 2^n*-Ausgänge angewählt werden.*

Anwendungen

Decodierer finden als Adressdecodierer Anwendung in digitalen Rechensystemen. Beispielsweise werden Peripheriegeräte oder Speicherzellen mit einer Adresse angewählt.



(a) Blockschaltbild

Eingang			Ausgang							
A_2	A_1	A_0	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7
0	0	0	**1**	0	0	0	0	0	0	0
0	0	1	0	**1**	0	0	0	0	0	0
0	1	0	0	0	**1**	0	0	0	0	0
0	1	1	0	0	0	**1**	0	0	0	0
1	0	0	0	0	0	0	**1**	0	0	0
1	0	1	0	0	0	0	0	**1**	0	0
1	1	0	0	0	0	0	0	0	**1**	0
1	1	1	0	0	0	0	0	0	0	**1**

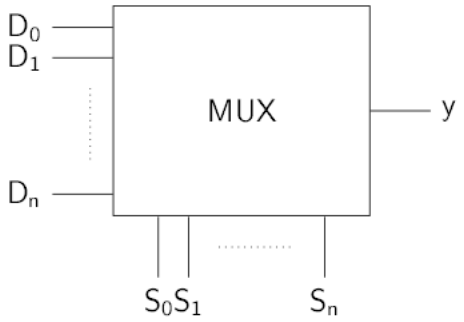
(b) Wertetabelle

Abbildung 5.4: 3-Bit Adressdecodierer

5.3 Multiplexer

Funktion

Ein Multiplexer ist ein **auswählendes** Schaltnetz. Über Steuereingänge wird einer der Dateneingänge auf den Ausgang durchgeschaltet.



(a) Blockschaltbild

$D_0, ..., D_n$	Dateneingänge
$S_0, ..., S_m$	Steuereingänge
Y	Ausgang
G	Steuereingänge steuern Dateneingänge durch UND-Verknüpfung
$\overline{E}$	Aktivierungseingang Enable (**Low-aktiv**)

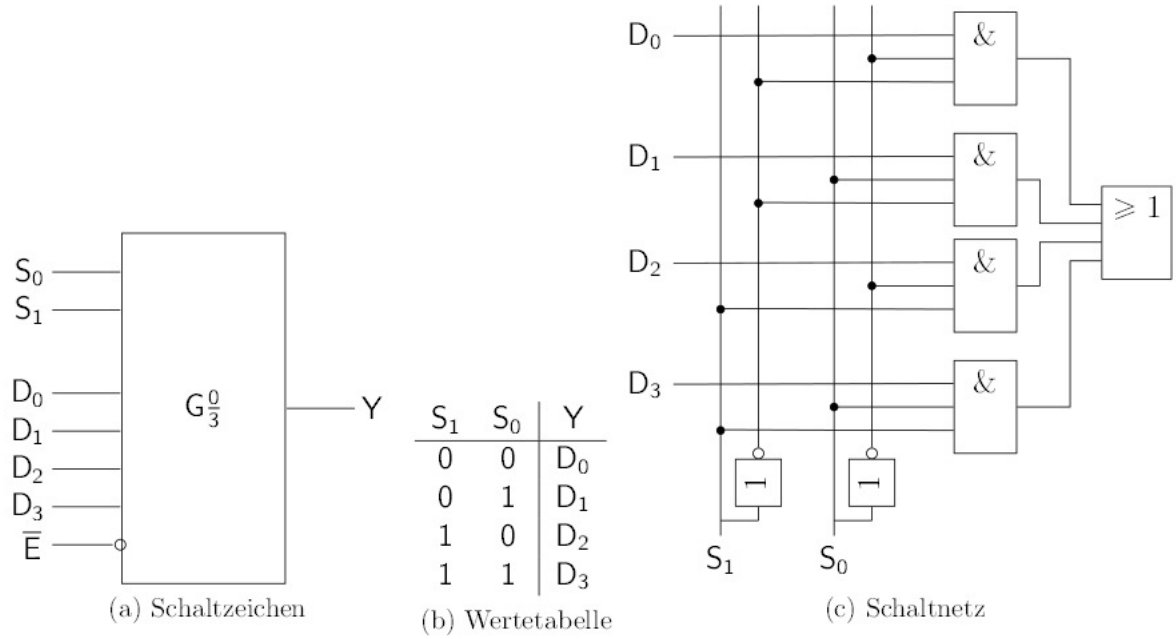
(b) Ein- und Ausgänge

Abbildung 5.5: Multiplexer

Beispiel 68. *4-zu-1 Multiplexer*
Die Funktion eines 4-zu-1 Multiplexers wird durch eine Wertetabelle beschrieben.

Logikfunktion
Jeweils ein Dateneingang wird mit dem entsprechenden Steuerwort UND-verknüpft und auf den Ausgang geschaltet.

$$Y = \overline{S_0}\ \overline{S_1}\ D_0 + S_0\ \overline{S_1}\ D_1 + \overline{S_0}\ S_1\ D_2 + S_0\ S_1\ D_3$$



(a) Schaltzeichen (b) Wertetabelle (c) Schaltnetz

Abbildung 5.6: 4-zu-1 Multiplexer

Schaltnetz
Aus der Logikfunktion ergibt sich das Schaltnetz (siehe Abbildung).

Multiplexer mit größerer Wortbreite

Typische Datenwortlängen, die in einem Rechenwerk verarbeitet werden, sind 4 bis 64 Bits. Ein Multiplexer, der Datenworte von einem auswärtigen Register auf die ALU durchschaltet, muss auch 4 bis 64 Bit-Eingangsdaten auf den Ausgang mit entsprechender Wortlänge schalten.

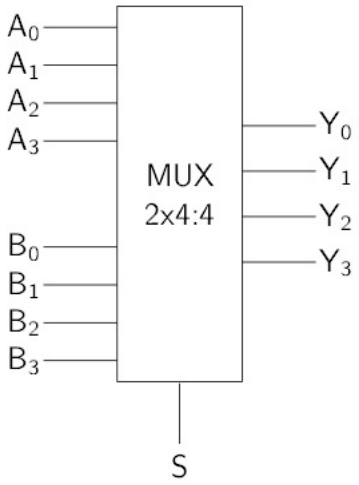
Beispiel 69. *2x4-zu-4 Multiplexer*

Es werden die Dateneingänge $A_0, .., A_3$ *oder* $B_0, .., B_3$ *auf den Ausgang* Y *durchgeschaltet. Die Funktion des 2x4-Bit zu 4-Bit Multiplexers wird durch die Wertetabelle beschrieben. Die Auswahl der Eingangsdatenworte wird durch die Steuervariable S festgelegt.*

5.4 Demultiplexer

Funktion

Der Demultiplexer ist ein **verteilendes** Schaltnetz. In Abhängigkeit vom Steuerwort wird ein Dateneingang auf einen der möglichen Datenausgänge geschaltet. Mit n Steuereingängen kann auf einen von 2^n Datenausgängen verteilt werden.

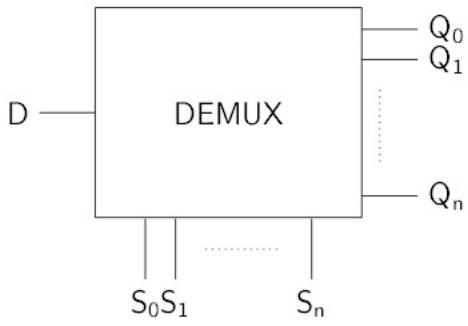


(a) Blockschaltbild

S	Y
0	$A_{0...3}$
1	$B_{0...3}$

(b) Wertetabelle

Abbildung 5.7: 2x4-zu-4 Multiplexers



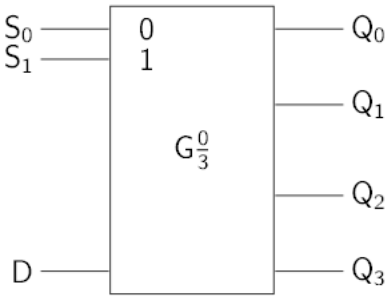
(a) Blockschaltbild

D	Dateneingang
$S_0, ..., S_m$	Steuereingänge
$Q_0, ..., Q_n$	Ausgänge
G	Steuereingänge steuern Dateneingang durch UND-Verknüpfung

(b) Ein- und Ausgänge

Abbildung 5.8: Demultiplexer

Beispiel 70. *1-zu-4 Demultiplexer*



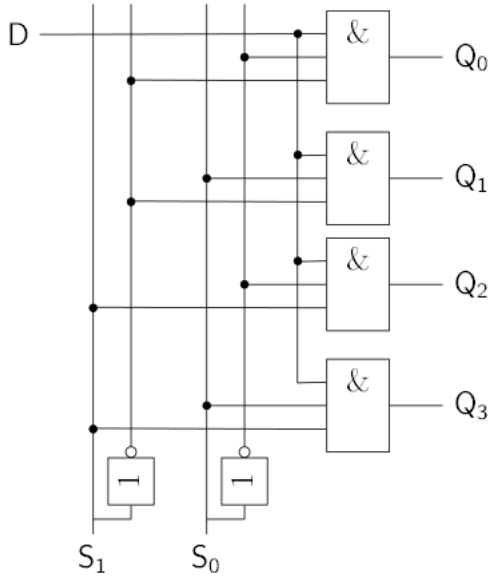
(a) Schaltzeichen

S_1	S_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

(b) Wertetabelle

$$Q_0 = \overline{S_0}\ \overline{S_1}\ D$$
$$Q_2 = \overline{S_0}\ S_1\ D$$
$$Q_1 = S_0\ \overline{S_1}\ D$$
$$Q_3 = S_0\ S_1\ D$$

(c) Schaltfunktionen



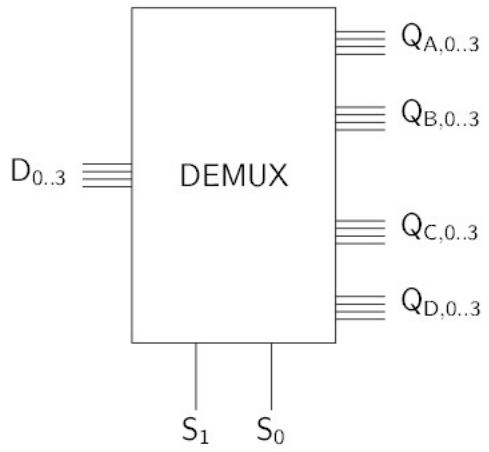
(d) Schaltnetz

Abbildung 5.9: 1-zu-4 Demultiplexer

Die Steuerworteingänge und der Dateneingang sind wie beim Multiplexer UND-verknüpft. Aus der Wertetabelle ergeben sich für die Ausgänge die entsprechenden Schaltfunktionen.

Mehrfach-Demultiplexer

Um ein Mehr-Bit Dateneingangswort $D_{0...3}$ auf verschiedene Mehr-Bit Ausgangskanäle $Q_{A,...,D}$ schalten zu können, müssen entsprechende Demultiplexer vorhanden sein. Die Funktion eines 1x4-Bit zu 4x4-Bit Demultiplexers wird durch folgende Wertetabelle beschrieben:



(a) Blockschaltbild

S_1	S_0	$Q_{A_{0...3}}$	$Q_{B_{0...3}}$	$Q_{C_{0...3}}$	$Q_{D_{0...3}}$
0	0	$D_{0...3}$	0	0	0
0	1	0	$D_{0...3}$	0	0
1	0	0	0	$D_{0...3}$	0
1	1	0	0	0	$D_{0...3}$

(b) Wertetabelle

Abbildung 5.10: 1x4-zu-4x4 Demultiplexers

5.5 Komparatoren

Funktion

Komparatoren **vergleichen** analoge oder binäre Signale. In Digitalsystemen sind Komparatoren Schaltnetze, die zwei Binärzahlen miteinander vergleichen. Für zwei Binärzahlen a und b gelten z. B. die Vergleichskriterien **a=b**, **a<b** und **a>b** .

Komparatorschaltnetz für zwei einstellige Binärzahlen

Aus der Wertetabelle können die Schaltfunktionen direkt angegeben werden (siehe Abbildung 5.11).

Komparatorschaltnetz für zwei zweistellige Binärzahlen

Sollen mehrstellige Binärzahlen auf $=$, $>$ oder $<$ untersucht werden, dann sind verschiedene Schaltungskonfigurationen denkbar. Mithilfe der Stellenwertigkeit der beiden Binärzahlen kann der Vergleich in einer Wertetabelle dargestellt werden. Aus der Wertetabelle wiederum ergeben sich nach Vereinfachung die entsprechenden Schaltfunktionen, die dann als Schaltnetz realisiert werden können.

Beispiel 71. *2-Bit Komparator*

Für zwei zweistellige Binärzahlen $A = a_1a_0$ *und* $B = b_1b_0$*, wobei* a_0, b_0 *den Stellenwert* 2^0 *haben, und* a_1, b_1 *den Stellenwert* 2^1*, ergibt sich die Wertetabelle:*

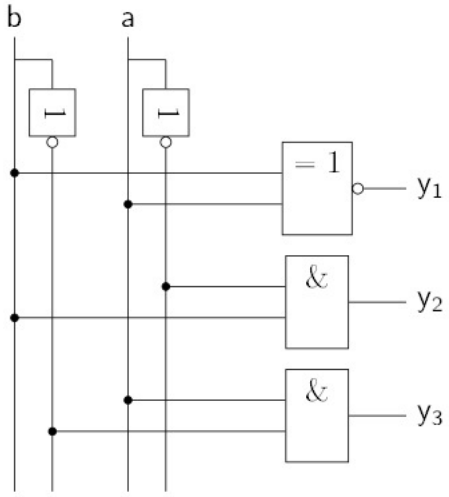
(a) Blockschaltbild

b	a	y_1 a = b	y_2 a>b	y_3 a<b
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

(b) Wertetabelle

$y_1 = a \equiv b$
$y_2 = \overline{a}\ b$
$y_3 = a\ \overline{b}$

(c) Schaltfunktionen



(d) Schaltnetz

Abbildung 5.11: 1-Bit Komparator

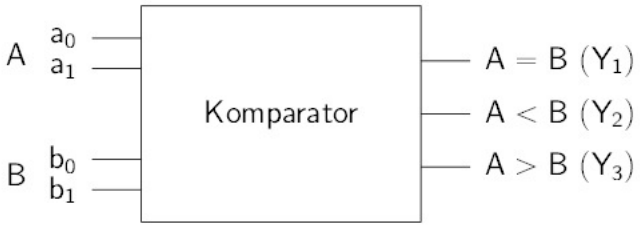


Abbildung 5.12: 2-Bit Komparator

B		A		Y_1	Y_2	Y_3
b_1	b_0	a_1	a_0	A = B	A > B	A < B
0	0	0	0	1	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	0	0	1
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	0	1	0
1	1	1	1	1	0	0

Aus der Wertetabelle ergeben sich nach Vereinfachung die Schaltfunktionen in der DNF:

$Y_2 = \overline{a_1}\ b_1 + \overline{a_1}\ \overline{a_0}\ b_0 + \overline{a_0}\ b_1\ b_0$ *für* A < B

$Y_3 = a_1\ \overline{b_1} + a_0\ \overline{b_1}\ \overline{b_0} + a_1\ a_0\ \overline{b_0}$ *für* A > B

$Y_1 = \overline{Y_2}\ \overline{Y_3} = \overline{Y_2 + Y_3}$ *für* A = B

Diese Schaltfunktionen können als Schaltnetz realisiert werden.

Vergleich mehrstelliger Binärzahlen

Es wird ein Algorithmus angewandt, der schrittweise alle Bit-Stellen miteinander vergleicht. Der Vergleich kann mit der MSB-Stelle (werthöchste) oder der LSB-Stelle (wertniedrigste) beginnen. Die Realisierung führt zu **mehrstufigen** Schaltnetzen.

5.6 Arithmetische Schaltnetze (Addierer)

In einem Computer gehören Addierer zur arithmetisch/logischen Einheit (engl.: Arithmetic Logic Unit [ALU]). Addierer sind Schaltnetze, die zwei Dualzahlen miteinander

addieren. Dualzahlen werden wie Dezimalzahlen stellenweise addiert, beginnend bei der niederwertigsten Stelle (least significant bit [LSB]).

5.6.1 Halbaddierer

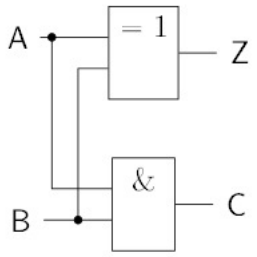
Die Addition zweier einstelliger Dualzahlen A und B ergibt vier Wertekombinationen. Das Ergebnis wird mit **Summe Z** und **Übertrag C** gekennzeichnet. Diese vier Kombinationen können in eine Wertetabelle übertragen werden, aus der man die Schaltfunktionen für Z und C in eine DNF übertragen kann.

A	B	Z	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

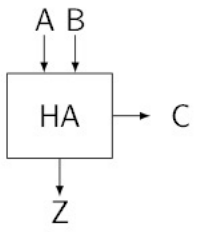
(a) Wertetabelle

$$Z = A\,\overline{B} + \overline{A}\,B = A \oplus B$$
$$C = A\,B$$

(b) Schaltfunktionen



(c) Schaltnetz



(d) Schaltsymbol

Abbildung 5.13: Halbaddierer

Überträgt man diese Funktion in ein Schaltnetz, erhält man die Gatterstruktur für einen einfachen Addierer. Betrachtet man nun den Addierer als ein Gatter mit den zwei einstelligen Eingängen A und B und den Ausgängen Z und C, so erhält man einen **Halbaddierer (HA)**.

5.6.2 Volladdierer

Bei Addition von zwei **mehrstelligen** Dualzahlen (A und B) werden nicht zwei sondern drei Bit addiert, weil der Übertrag (C_i) von der nächst niedrigeren Stelle hinzukommt. Ein Schaltnetz, das drei Bit addieren kann und daraus die Summe Z und den Übertrag C_{i+1} bildet, wird **Volladdierer (VA)** genannt. Solche Additionen können nicht mit einem Halbaddierer durchgeführt werden.

Auch für den Volladdierer gibt es eine Wertetabelle und Schaltfunktion, die zu einem wesentlich komplexeren internen Aufbau des Volladdierers im Vergleich zum Halbaddierer führen.

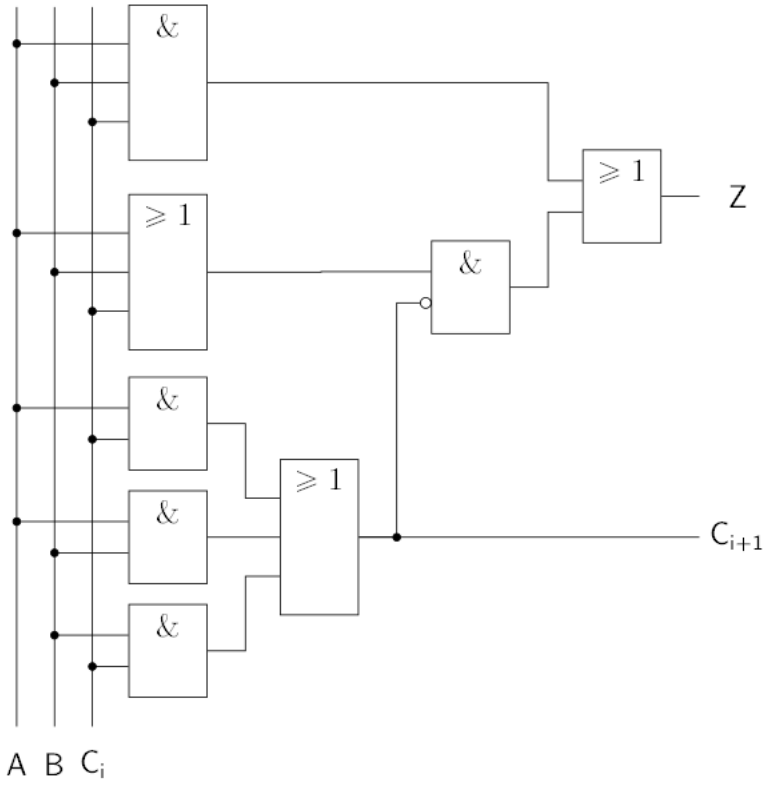
C_i	A	B	Z	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) Wertetabelle

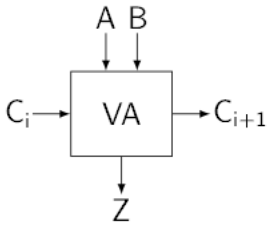
$$Z = A\,\overline{B}\,\overline{C_i} + \overline{A}\,B\,\overline{C_i} + \overline{A}\,\overline{B}\,C_i + A\,B\,C_i$$
$$Z = A \oplus B \oplus C_i$$

$$C_{i+1} = A\,B + A\,C_i + B\,C_i$$
$$C_{i+1} = A\,B + (A + B)C_i$$

(b) Schaltfunktionen



(c) Schaltnetz



(d) Schaltsymbol

Abbildung 5.14: Volladdierer

5.6.3 Mehrstellige Addierer

Addition zweier mehrstelliger Dualzahlen kann bitseriell oder bitparallel erfolgen. Man spricht daher auch von **Serienaddierer** und **Paralleladdierer**. Beide Addiernetze unterscheiden sich wesentlich im Hardwareaufwand und in der Addierzeit. Serienaddierer führen pro Taktschritt nur **eine Stelle** der Addition aus (mit Speicherelementen). Paralleladdierer hingegen führen während eines Taktschrittes die Addition **aller Stellen** aus.

5.7 Arithmetisch/Logische Einheit (ALU)

Die Basisfunktionalität von programmgesteuerten datenverarbeitenden Geräten wird von arithmetisch-logischen Einheiten abgebildet. Sie stellen somit den Kern des datenverarbeitenden Operationswerks dar. Ihre Komplexität ist hoch genug um mehrere elementare Operationen durchführen zu können. Diese lassen sich - wie man dem Namen der Einheit entnehmen kann - in arithmetische und logische Operationen gruppieren. Trotz der Vielzahl an durchführbaren Operationen ist die strukturelle Komplexität (der Aufbau) einer ALU, bedingt durch die Ähnlichkeiten zwischen den einzelnen Operationen, relativ gering.

Der wesentliche Unterschied zwischen den arithmetischen und logischen Operationen besteht darin, dass erstere auf 2-Komplement-Zahlen arbeiten, letztere aber auf einzelnen Bits, wobei diese jedoch zu Bitvektoren zusammengefasst werden können.

Es können alle 16 logischen Operationen (siehe Tabelle 1.1) durchgeführt werden, die mit zwei Variablen möglich sind. Bei den arithmetischen Operationen sind nur elementare realisierbar (Zählen, Komplementieren, Addieren, Subtrahieren). Komplexere Operationen (Multiplizieren, Dividieren, Radizieren) werden durch zusätzliche Programme, Schaltwerke oder Schaltnetze abgebildet.

Herleiten der ALU-Struktur

Bei der Verarbeitung von 2-Komplement-Zahlen ist es wichtig, Strukturen vorzusehen, die den Austausch von Informationen über die einzelnen Bitstellen hinweg ermöglichen. Bei dem Volladdierer ist eine solche Struktur mit dem Übertrags-Bit vorhanden. Mit Hilfe der Volladdiererergleichung und einigen Umformungen lässt sich aus einem Volladdiererglied ein ALU-Glied ableiten.

Die Volladdiererergleichungen:

$$\begin{array}{ll} c_{i+1} = a_i \cdot b_i + (a_i + b_i) \cdot c_i & \text{Übertragungsfunktion} \\ z_i \quad = a_i \equiv b_i \equiv c_i & \text{Summenfunktion} \end{array}$$

lassen sich durch folgende Ersetzungen:

$$\begin{array}{ll} G_i = a_i \cdot b_i & \text{Generate (Übertrag generieren)} \\ P_i = a_i + b_i & \text{Propagate (Übertrag propagieren)} \end{array}$$

umschreiben zu:

$$\begin{aligned} c_{i+1} &= G_i + P_i \cdot c_i \\ z_i &= a_i \equiv b_i \equiv c_i \\ &= (a_i \cdot b_i + \overline{a_i + b_i}) \equiv c_i \\ &= (G_i + \overline{P_i}) \equiv c_i \end{aligned}$$

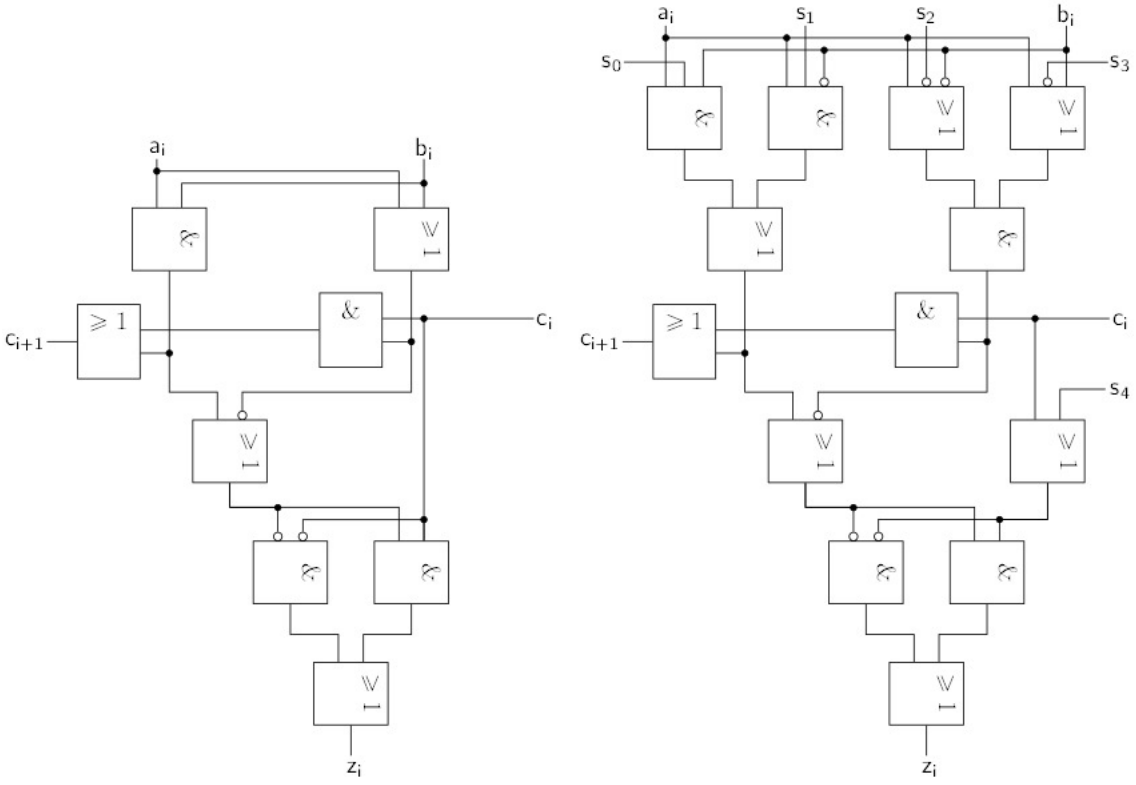
Aus dem einfachen 1-Bit-Volladdierer entsteht eine 1-Bit-ALU, indem die so entstandenen Gleichungen um einen Steuervektor s erweitert wird. Dieser dient dazu die Generate- und Propagate-Funktions-Variable erzeugen zu können sowie die Funktionalität um logische Operationen zu erweitern.

Mit dem Steuervektor $s = [s_0 s_1 s_2 s_3 s_4]$ lauten die Gleichungen des ALU-Glieds:

$$\begin{aligned} G_i &= s_0 \cdot a_i \cdot b_i + s_1 \cdot a_i \cdot \overline{b_i} \\ P_i &= (\overline{s_2} + a_i + \overline{b_i}) \cdot (\overline{s_3} + a_i + b_i) \\ c_{i+1} &= G_i + P_i \cdot c_i \\ z_i &= (G_i + \overline{P_i}) \equiv (s_4 + c_i) \end{aligned}$$

Durch Variation von $s_0 \ldots s_4$ können zusätzlich zur Addition (Steuervektor $s = [10010]$) eine Vielzahl weiterer Operationen gebildet werden. s_4 dient dazu zwischen den beiden Operationsgruppen zu unterscheiden. Mit $s_4 = 0$ ist das Ergebnis z_i abhängig vom Übertrag c_i, weil sich die Gleichung wieder auf $z_i = (G_i + \overline{P_i}) \equiv c_i$ reduzieren lässt. $s_4 = 0$ kennzeichnet somit die arithmetischen Operationen. Demzufolge werden die logischen Operation durch $s_4 = 1$ gekennzeichnet, denn dadurch wird das Ergebnis $(z_i = G_i + \overline{P_i})$ völlig unabhängig vom Übertrag c_i.

Sämtliche ALU-Operationen mit n-stelligen 2-Komplement-Zahlen bzw. Bitvektoren lassen sich der Tabelle 5.1 entnehmen.



(a) VA-Glied (b) ALU-Glied

Abbildung 5.15: Gegenüberstellung eines Volladdierer-Gliedes mit einem ALU-Glied

s_0	s_1	s_2	s_3	arithmetische Operationen	logische Operationen
0	0	0	0	$Z = \quad 1 + c_0$	$Z = 0$
0	0	0	1	$Z = A \vee B + c_0$	$Z = \overline{A \vee B}$
0	0	1	0	$Z = A \vee \overline{B} + c_0$	$Z = \overline{A} \wedge B$
0	0	1	1	$Z = A + c_0$	$Z = \overline{A}$
0	1	0	0	$Z = A \wedge \overline{B} \quad 1 + c_0$	$Z = A \wedge \overline{B}$
0	1	0	1	$Z = (A \vee B) + (A \wedge \overline{B}) + c_0$	$Z = \overline{B}$
0	1	1	0	$Z = A \quad B \quad 1 + c_0$	$Z = A \not\equiv B$
0	1	1	1	$Z = (A \wedge \overline{B}) + A + c_0$	$Z = \overline{A \wedge B}$
1	0	0	0	$Z = (A \vee B) \quad 1 + c_0$	$Z = A \wedge B$
1	0	0	1	$Z = A + B + c_0$	$Z = A \equiv B$
1	0	1	0	$Z = (A \vee \overline{B}) + (A \wedge \overline{B}) + c_0$	$Z = B$
1	0	1	1	$Z = (A \wedge B) + A + c_0$	$Z = \overline{A} \vee B$
1	1	0	0	$Z = A \quad 1 + c_0$	$Z = A$
1	1	0	1	$Z = (A \vee B) + A + c_0$	$Z = A \vee \overline{B}$
1	1	1	0	$Z = (A \vee \overline{B}) + A + c_0$	$Z = A \vee B$
1	1	1	1	$Z = A \times 2 + c_0$	$Z = \quad 1$

Tabelle 5.1: Operationen einer ALU. Anmerkung: logische Operationen werden hier mit $\vee$ und $\wedge$ dargestellt, um sie von den arithmetischen (Addition $+$, Multiplikation $\times$) unterscheiden zu können.

5.8 Schaltketten

Schaltketten sind mehrstufige Schaltnetze, die vorwiegend aus identischen bzw. generischen Teilschaltnetzen aufgebaut sind. Sie treten bei der Realisierung rekursiver Funktionen auf. Charakteristisch für solche Funktionen ist, dass auf beiden Seiten der Funktionsgleichung Variablen mit gleichem Namen (nur durch einen Index unterschieden) auftreten. Zum Beispiel:

$$w_{i+1} = f(a_i, b_i, w_i), \text{ für } i = 0, 1, \ldots, n \quad 1$$

Dies gilt als Kurzschreibweise für

$$w_n = f(a_{n\ 1}, b_{n\ 1}, f(a_{n\ 2}, b_{n\ 2}, \ldots, f(a_0, b_0, w_0))) \text{ mit } w_0 = 1 \text{ oder } w_0 = 0$$

Mit rekursiven Funktionen lassen sich Operationen zwischen Dualzahlen (Addition, Multiplikation, Vergleich) gut beschreiben. Derartige Schaltungen haben Bedeutung in Rechenanlagen. Ein Typisches Beispiel für rekursive Schaltfunktionen ist die Übertragsfunktion bei der Addition. Eine Verknüpfung von Volladdierern zum Zwecke der Addition zweier n-Stelliger Zahlen ergibt dabei eine Schaltkette. Wir vereinbaren dazu, dass Dualziffern ohne Umbenennung als Schaltvariablen behandelt werden.

Beispiel 72. *Additionsschaltkette für zwei Dualzahlen*
Mit Volladdierern als Zelle wird ein Addierer für zwei Dualzahlen $a = a_n a_{n\ 1} \ldots a_0$, $b = b_n b_{n\ 1} \ldots b_0$ *als Schaltkette folgendermaßen realisiert:*

$$c_{i+1} = a_i \cdot b_i + a_i \cdot c_i + b_i \cdot c_i = F(a_i, b_i, c_i) \qquad \textit{Übertragsfunktion}$$
$$z_i = (a_i \oplus b_i) \oplus c_i = G(a_i, b_i, c_i) \qquad \textit{Summenfunktion}$$

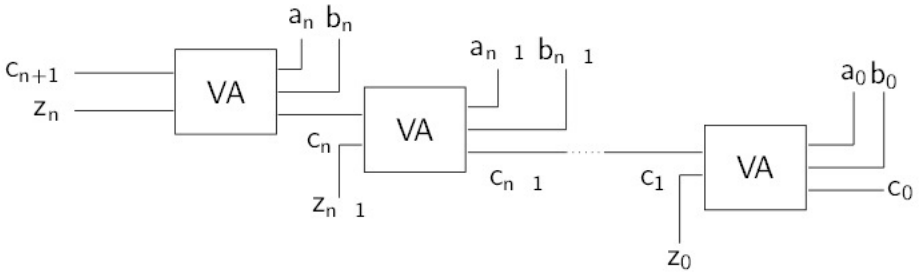


Abbildung 5.16: Additionsschaltkette für zwei Dualzahlen

Die Glieder einer Schaltkette werden auch Kettenglieder genannt. Sie haben stets das gleiche Schaltverhalten und verarbeiten neben Eingangsvariablen a_i, b_i auch Überträge c_i vom vorherigen Kettenglied, die durch die Übergangsfunktion F bestimmt sind. Die Überträge für das 1. Kettenglied ($c_0 = 0$) werden als Anfangswerte bezeichnet. Da Anfangswerte meist Schaltkonstanten sind, ist das 1. Glied der Kette häufig eine einfachere Schaltung. Jedes Kettenglied kann außerdem Ausgangsvariablen haben (hier z_i). Die dazugehörige Schaltfunktion (das zugehörige Funktionsbündel) heißt Ausgangsfunktion (G). Auch die Übergangsfunktion kann ein Funktionsbündel sein.

Beispiel 73. *Eindimensionale Schaltkette*

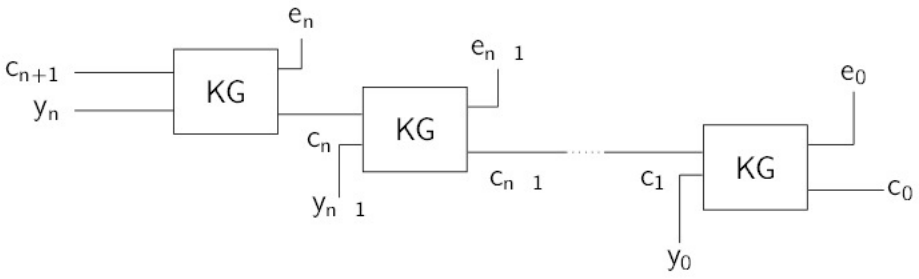


Abbildung 5.17: Verallgemeinerte eindimensionale Schaltkette

Es werden die folgenden n*-Tupel von Schaltvariablen gebildet:*

$$
\begin{aligned}
e_i &= (e_{1i}, e_{2i}, \ldots, e_{ri}) && \textit{Eingangsvektor}\\
c_i &= (c_{1i}, c_{2i}, \ldots, c_{ri}) && \textit{Übergangsvektor}\\
y_i &= (y_{1i}, y_{2i}, \ldots, y_{ri}) && \textit{Ausgangsvektor}\\
c_{i+1} &= F(e_i, c_i) && \textit{Übergangsfunktion}\\
y_i &= G(e_i, c_i) && \textit{Ausgangsfunktion}\\
c_0 & && \textit{Anfangsvektor}
\end{aligned}
$$

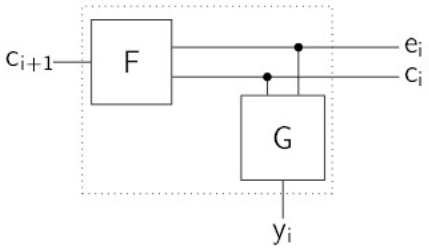


Abbildung 5.18: Kettenglied

Jedes Kettenglied kann selbst ein Schaltnetz sein. Beim Entwurf von Schaltketten ist zunächst eine schaltalgebraische Beschreibung des Problems mit rekursiver Übergangsfunktion und Ausgangsfunktion aufzustellen. Danach ist das Kettenglied als Schaltnetz zu entwerfen. Es ist jedoch nicht jedes schaltalgebraische Problem dazu geeignet, mit Schaltketten realisiert zu werden.

Beispiel 74. *Komparator für zwei Dualzahlen*
Entwurf einer Schaltkette, die zwei $n+1$*-stellige Dualzahlen auf Gleichheit prüft:*

$$
\begin{aligned}
a &= a_n a_{n\ 1} \ldots a_0\\
b &= b_n b_{n\ 1} \ldots b_0\\
w &= \begin{cases} 1, & \textit{für } a = b\\ 0, & \textit{für } a \neq b \end{cases}
\end{aligned}
$$

Zur Vereinfachung werden Äquivalenzgatter verwendet. Gleichheit tritt genau dann ein, wenn die Dualziffern an jeder Stelle paarweise gleich sind:

$$w = (a_n \equiv b_n) \cdot (a_{n\ 1} \equiv b_{n\ 1}) \cdot \ldots \cdot (a_0 \equiv b_0)$$

Dieser logische Ausdruck kann rekursiv abgearbeitet werden:

$$
\begin{aligned}
w_1 &= a_0 \equiv b_0 = (a_0 \equiv b_0) \cdot w_0 && \textit{mit } w_0 = 1\\
w_2 &= (a_1 \equiv b_1) \cdot w_1\\
w_{n+1} &= (a_n \equiv b_n) \cdot w_n\\
w &= w_{n+1}
\end{aligned}
$$

Die Übergangsfunktion ist:

$$w_{i+1} = F(a_i, b_i, w_i) = (a_i \equiv b_i) \cdot w_i \qquad i = 1, \ldots, n$$
$$w_1 = a_0 \equiv b_0$$

Das erste Kettenglied wurde mit dem Anfangswert $w_0 = 1$ *vereinfacht.*

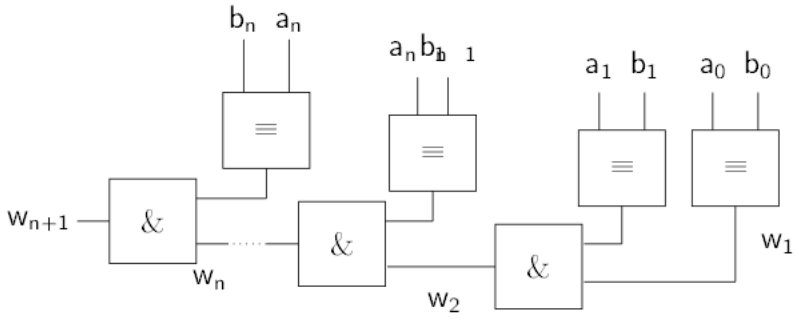


Abbildung 5.19: Komparator für zwei Dualzahlen

Laufzeitproblem

Schaltketten können beträchtliche Längen erreichen. Das bringt lange Signallaufzeiten und eventuell Hazards mit sich. Die Laufzeit ist das Produkt aus der Zahl der Kettenglieder und der Verzögerungszeit eines Gliedes. Die Nachteile sind nur teilweise durch schaltungstechnischen Mehraufwand auszugleichen. Ein 2-Stufiges Schaltnetz garantiert nicht die minimale Signallaufzeit. Es ist effizienter, einige Kettenglieder (meist 2, 4 oder 8) zusammenzufassen und 2-stufig zu realisieren. So werden beispielsweise 2-, 4- und 8-Bit-Volladdierer und 4- und 8-Bit-Vergleicher als Bausteine/Zellen eingesetzt, die sich dann wieder als Kettenglieder in Schaltketten verwenden lassen. Diese Bausteine sind allerdings nicht immer zweistufig realisiert.

Zwei- und mehrdimensionale Schaltketten

Bei zweidimensionalen Schaltketten empfängt jedes Kettenglied von zwei verschiedenen Kettengliedern Übergangssignale und gibt Signale an zwei Kettenglieder ab.

Beispiel 75. *Multiplikationsschaltkette*
Eine Schaltkette zur Multiplikation zweier 4-stelliger Dualzahlen

$$a = a_3a_2a_1a_0 \text{ } \textit{und} \text{ } b = b_3b_2b_1b_0$$

Die Multiplikation von Dualzahlen wird nach dem vom dezimalen Rechnen her geläufigen Algorithmus durchgeführt. Der Multiplikant wird mit jeder Ziffer des Multiplikators multipliziert. Das Endergebnis ergibt sich durch die stellenwertrichtige Aufsummierung der Teilprodukte.

$$
\begin{array}{cccccccccc}
 & & & & a_3 & a_2 & a_1 & a_0 & \cdot & b_0 \\
+ & & & a_3 & a_2 & a_1 & a_0 & & \cdot & b_1 \\
+ & & a_3 & a_2 & a_1 & a_0 & & & \cdot & b_2 \\
+ & a_3 & a_2 & a_1 & a_0 & & & & \cdot & b_3 \\
\hline
 & m_7 & m_6 & m_5 & m_4 & m_3 & m_2 & m_1 & m_0 & = & a \cdot b
\end{array}
$$

Es gilt:

$$
a_i \cdot b_i = \begin{cases} a, & \text{falls } b_i = 1 \\ 0, & \text{falls } b_i = 0 \end{cases}
$$

Um das Problem rekursiv formulieren zu können, erfolgt die Einführung der Zwischensummen:

$$
\begin{aligned}
z_i &= z_{i7} z_{i6} \ldots z_{i0} \\
z_0 &= 0000000 \\
z_{i+1} &= z_i + 2^i a b_i \qquad \text{für } i = 0, 1, 2, 3
\end{aligned}
$$

Damit gilt für das Produkt:

$$
z_4 = m_7 m_6 m_5 m_4 m_3 m_2 m_1 m_0 = a \cdot b
$$

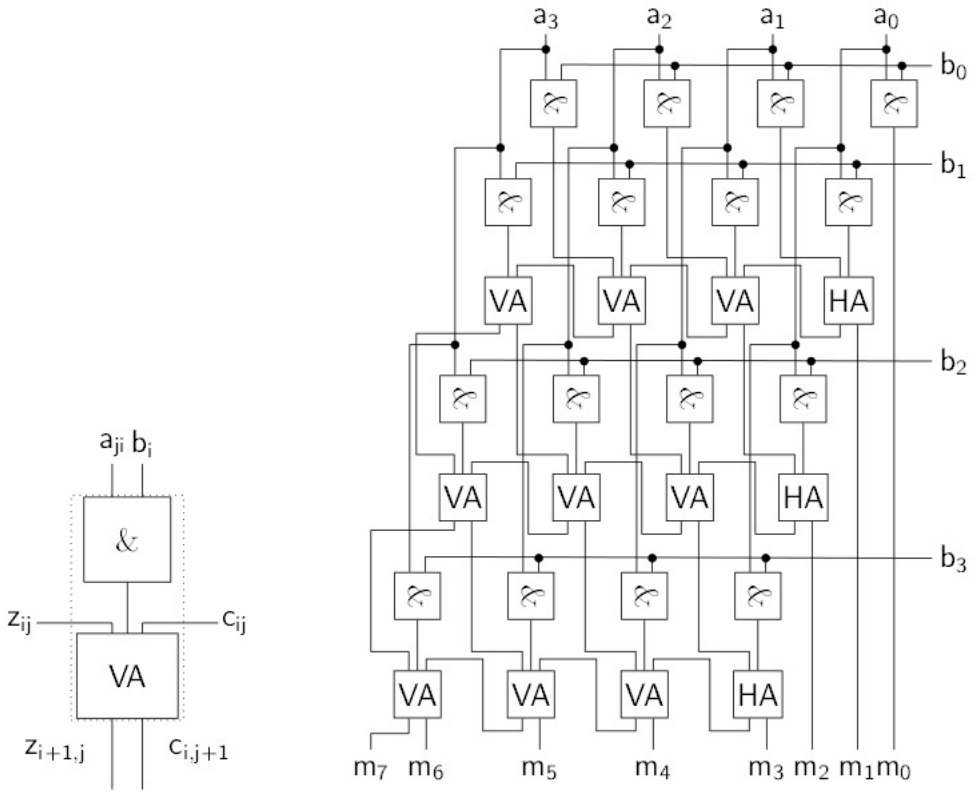
Für die Schaltalgebraische Beschreibung erhält man:

$$
\begin{aligned}
z_{i+1j} &= (z_{ij} \oplus (a_{j\ i} \cdot b_i)) \oplus c_{ij} \\
c_{ij+1} &= z_{ij} \cdot (a_{j\ i} \cdot b_i) + z_{ij} \cdot c_{ij} + (a_{j\ i} \cdot b_i) \cdot c_{ij} \\
z_{ii+4} &= c_{i\ 1,i+4} \\
\left.\begin{array}{ll} c_{ii} & = 0 \\ s_{0J} & = 0 \end{array}\right\} & \text{ Anfangswerte für } i = 0, 1, 2, 3 \text{ und } j = i, i+1, i+2, i+3
\end{aligned}
$$

Für die Dualziffern des Produktes $a \cdot b$ *gilt:*

$$
\begin{aligned}
m_j &= z_{j+1} \qquad \text{für } j = 0, 1, 2 \\
m_j &= z_{4j} \qquad \text{für } j = 3, 4, 5, 6 \\
m_7 &= c_{37}
\end{aligned}
$$

Die Beschreibung der Produktziffern wurde so gewählt, um unnötige Kettenglieder einzusparen. Statt 16 Kettengliedern würden sonst 28 benötigt, von denen 12 nur 0 addieren. Ein Kettenglied stellt sich dann wie in Abbildung 5.20a dar. Das in 5.20b dargestellte Schaltnetz wurde durch Ausnutzung der Anfangswerte vereinfacht.



(a) Kettenglied

(b) Multiplikationsschaltkette

Abbildung 5.20: Multiplikation zweier Dualzahlen

5.9 Schiebeeinheit

Eine weitere elementare Operation, neben den gezeigten arithmetsich-logischen Operationen, ist die Schiebeoperation (Shift). Auch wenn es unterschiedliche Ausprägungen von Schiebeoperationen gibt, so ist die prinzipielle Funktion immer die gleiche: bei einem Bitvektor wird jedes einzelne Bit von seiner ursprünglichen Stelle in diesem Vektor zu einer anderen verschoben.

Unterschieden werden die Schiebeoperationen zunächst nach der Richtung in welche geschoben wird und nach der Art wie mit den Bits am Anfang und Ende des Bitvektors umgegangen wird. Bits die sich an diesen Stellen befinden, werden durch die Schiebeoperation entweder aus dem Bereich des Vektors hinausgeschoben, oder Stellen müssen sinnvoll aufgefüllt werden. In Abbildung 5.21 sind die unterschiedlichen Varianten aufgeführt. Neben der Schieberichtung und -art kann man Shift-Einheiten auch noch um

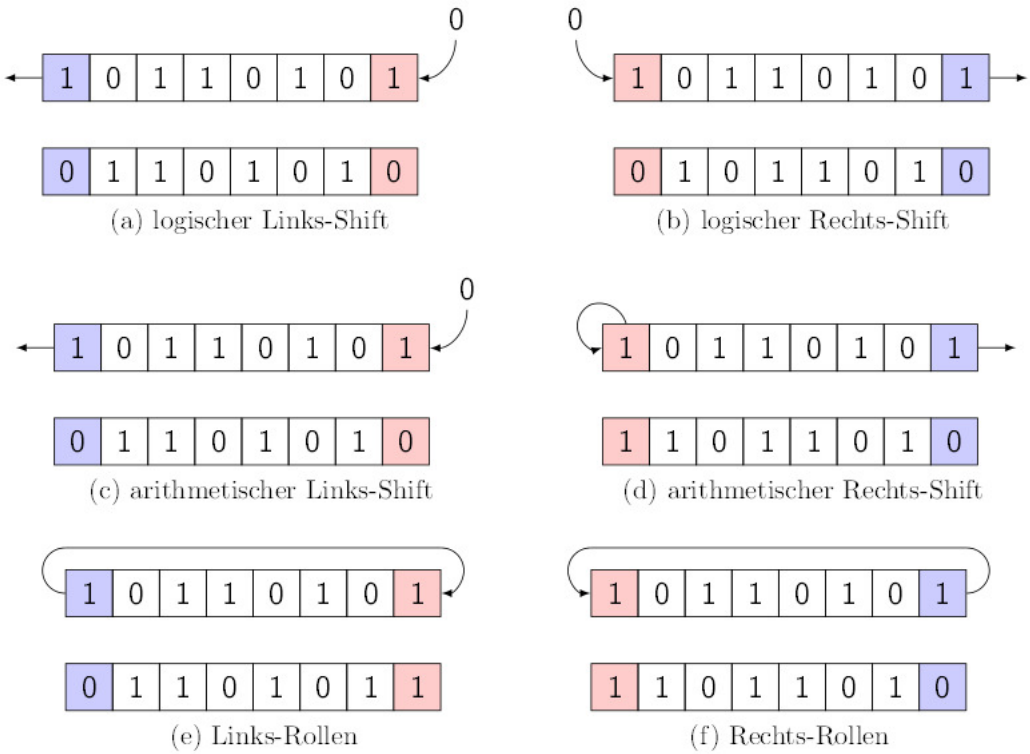


Abbildung 5.21: Die unterschiedlichen Ausprägungen der Schiebeoperation (Shift)

eine variable Shift-Weite ergänzen, die es ermöglicht einen Bitvektor um jede beliebige Stellenanzahl in jede Richtung zu verschieben. Schiebe-Operationen mit diesem Funktionsumfang sind in einer relativ einfachen ALU nicht realisierbar, da man beispielsweise den Übertrag eines Gliedes auch an die niederwertigen Glieder weitergeben können muss, um Schiebeoperationen nach rechts durchführen zu können. Deswegen werden Schiebeoperationen oft in extra Funktionseinheiten realisiert. Der bekannteste Vertreter einer Schiebeeinheit ist der Barrelshifter (siehe Abbildung 5.22).

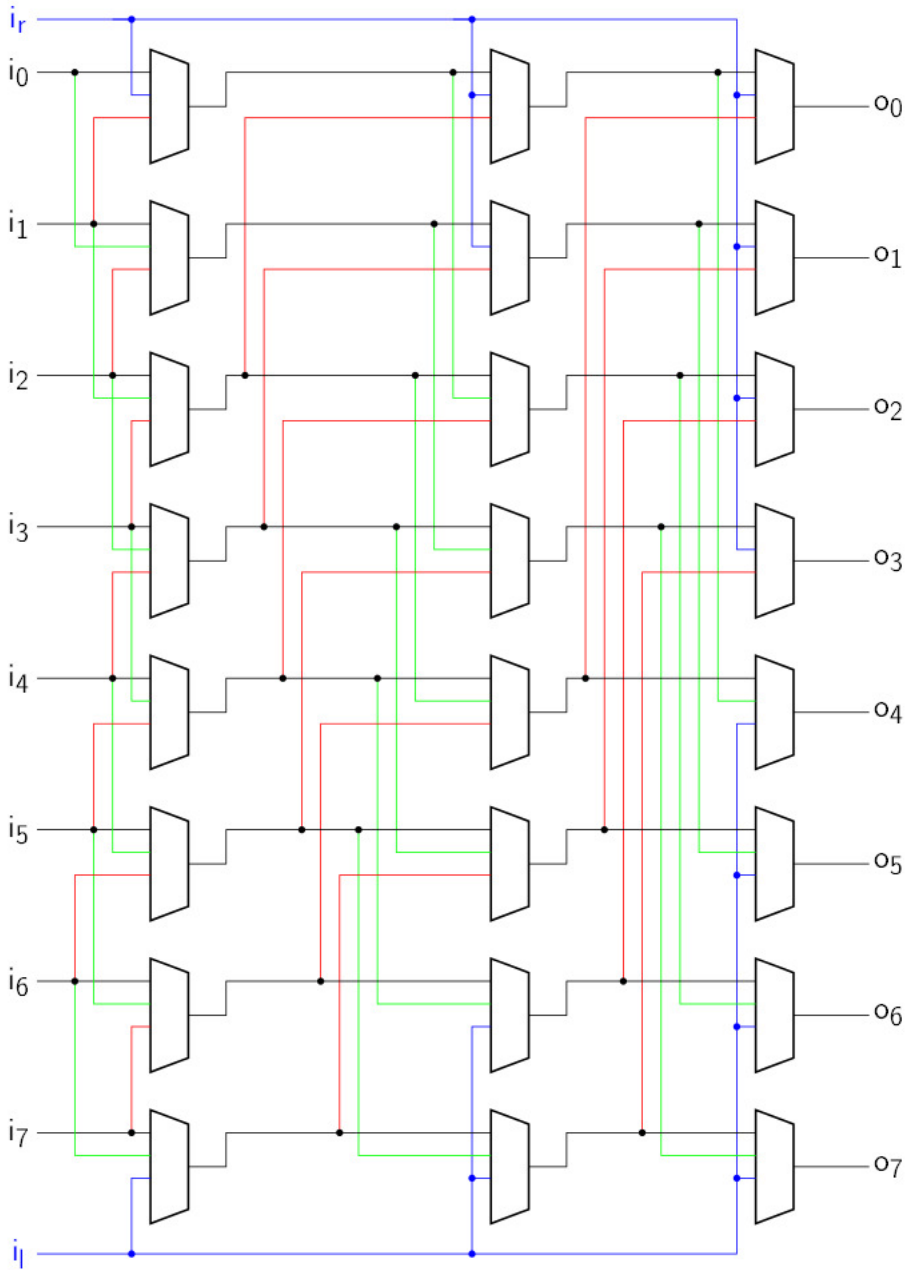


Abbildung 5.22: Multiplexerbasierte Implementierung eines Barrelshifters zur Realisierung von Schiebeoperationen mit variabler Schiebeweite