Lehrstuhl für
Eingebettete Systeme

Entwurfsmethodik
Institut für Informatik

Hardwarearchitekturen und Rechensysteme

4. Schaltnetze

Folien zur Vorlesung Hardwarearchitekturen und Rechensysteme von
Prof. Dr. rer. Nat. U. Brinkschulte
Prof. Dr.-Ing. L. Hedrich
Prof. Dr.-Ing. K. Waldschmidt

Motivation

Schaltnetze sind kombinatorische digitale Schaltungen. Rechnerwerkzeuge für den Entwurf und die Simulation dieser Schaltungen basieren auf der Booleschen Algebra.

In diesem Kapitel wird das Verhalten und die Struktur einiger ausgewählter Schaltnetze behandelt. Es sind Schaltnetze, die für den Aufbau von Operationswerken in Prozessoren benötigt werden.

Gliederung

4.1 Spezielle Schaltnetze

4.1.1 Multiplexer/Demultiplexer

4.1.2 Datenbuszugang

4.1.3 Permutationsschaltnetz

4.1.4 Vergleicher (Komparator)

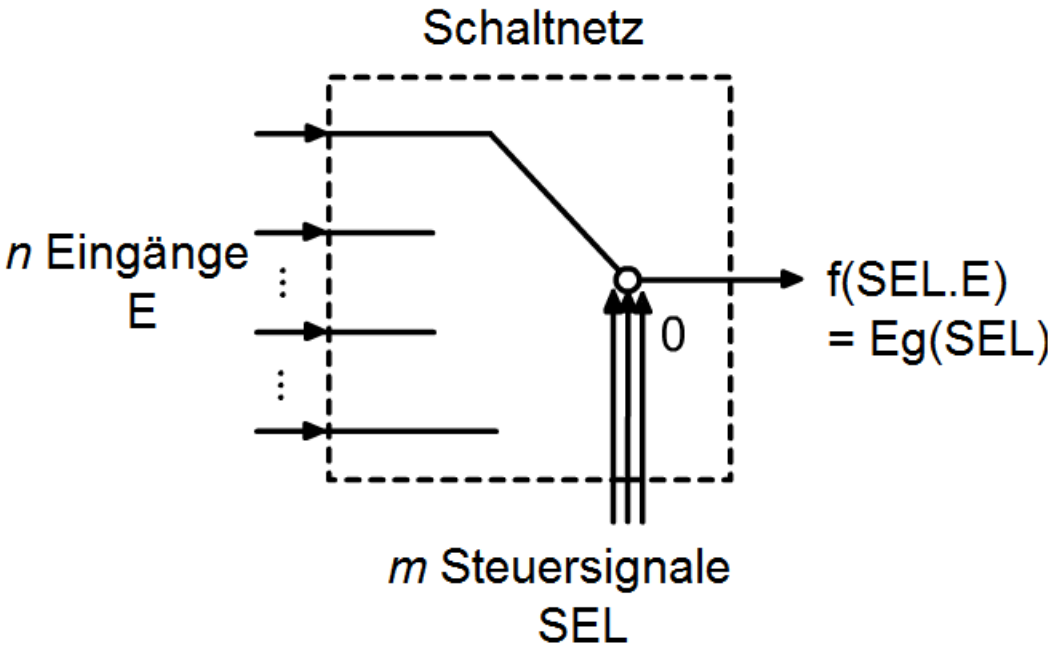
4.1.5 Addierer

4.1.6 Multiplizierer

4.1.1 Multiplexer/Demultiplexer

Ein Multiplexer/Demultiplexer ist ein Schaltnetz, welches eine Datenweiche darstellt.

Multiplexer:



Multiplexer

Definition:

Ein Multiplexer ist eine Boolesche Funktion f , für die gilt:

$$f : \mathbb{B}^m \times \mathbb{B}^n \rightarrow \mathbb{B} \qquad m \geq \lceil \log_2 n \rceil$$

$$f(SEL, E) = E_{g(SEL)} = E_0 \wedge (g == 0) \vee E_1 \wedge (g == 1)... \qquad SEL \in \mathbb{B}^m$$

$$= E_0 \wedge d_0 \vee E_1 \wedge d_1... \qquad E \in \mathbb{B}^n$$

Für die Abbildung g gilt:

$$g : \mathbb{B}^m \rightarrow \mathbb{N} \qquad g(SEL) = \sum_{i=0}^{m-1} sel_i * 2^i$$

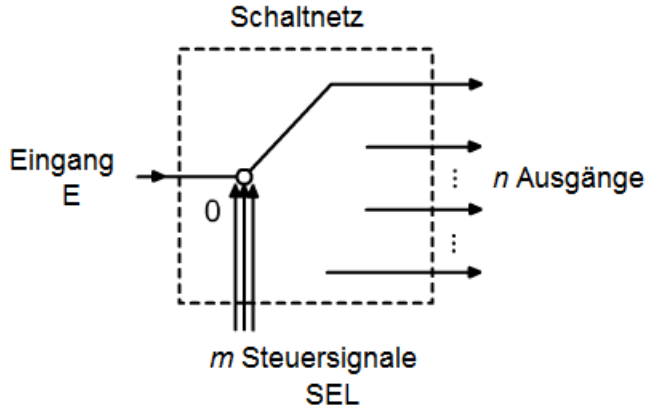
Für das Dekodiersignal d_k gilt:

$$d_k = (g == k)$$

Beispiel: $d_3 = (g == 3) = sel_0 \wedge sel_1 \wedge \overline{sel_2} \wedge \overline{sel_3}...$

Demultiplexer

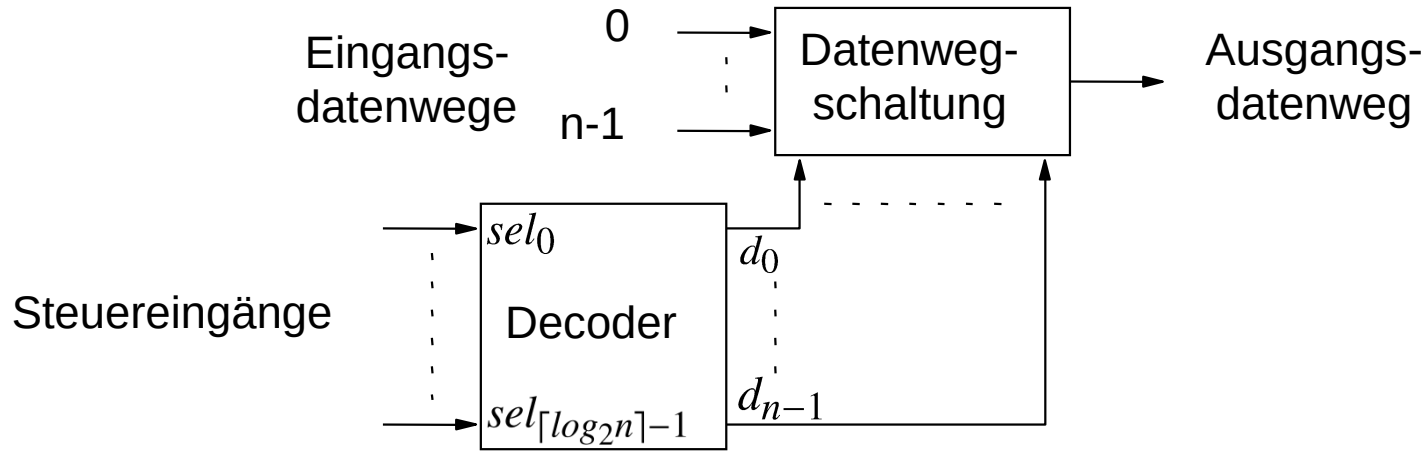
Demultiplexer arbeiten invers zu den Multiplexern. Sie verteilen einen Datenstrom auf mehrere auswählbare Kanäle.



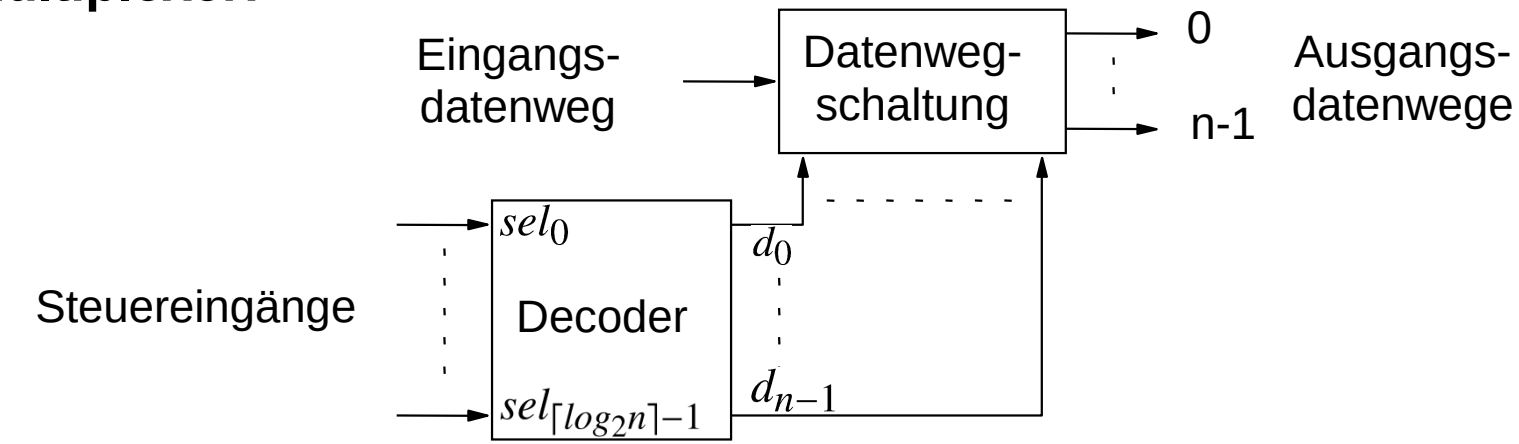
Demultiplexer finden oftmals in integrierten Schaltkreisen Anwendung, um die Zahl der Anschlußpins zu begrenzen. In DRAMs beispielsweise wird der höherwertige und der niederwertige Teil der Adresse nacheinander auf den Adreßbus gelegt. Der Baustein muß dann die Signale intern demultiplexen und dem Spalten- bzw. Zeilendekoder zuführen.

Datenwegschaltung

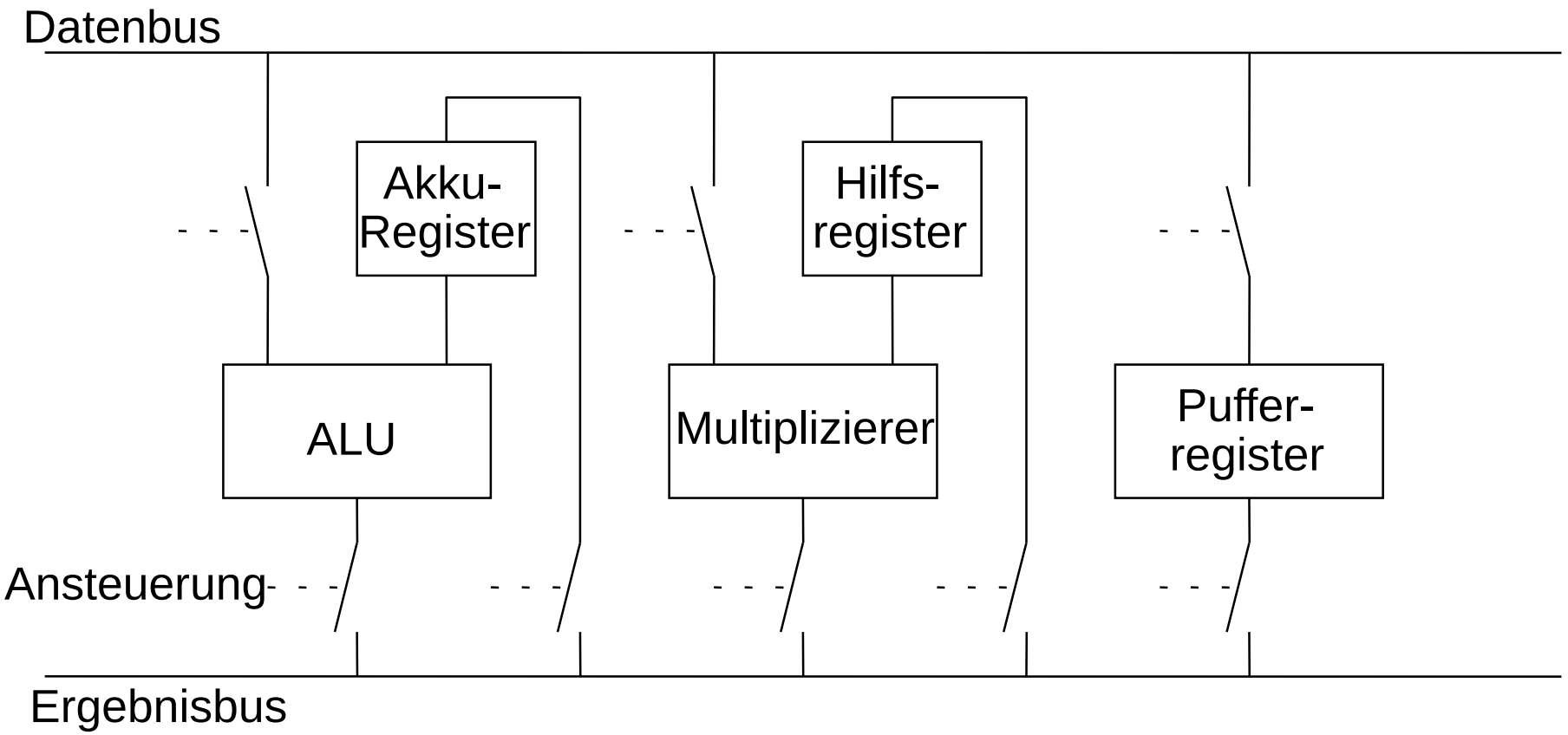
Multiplexer:



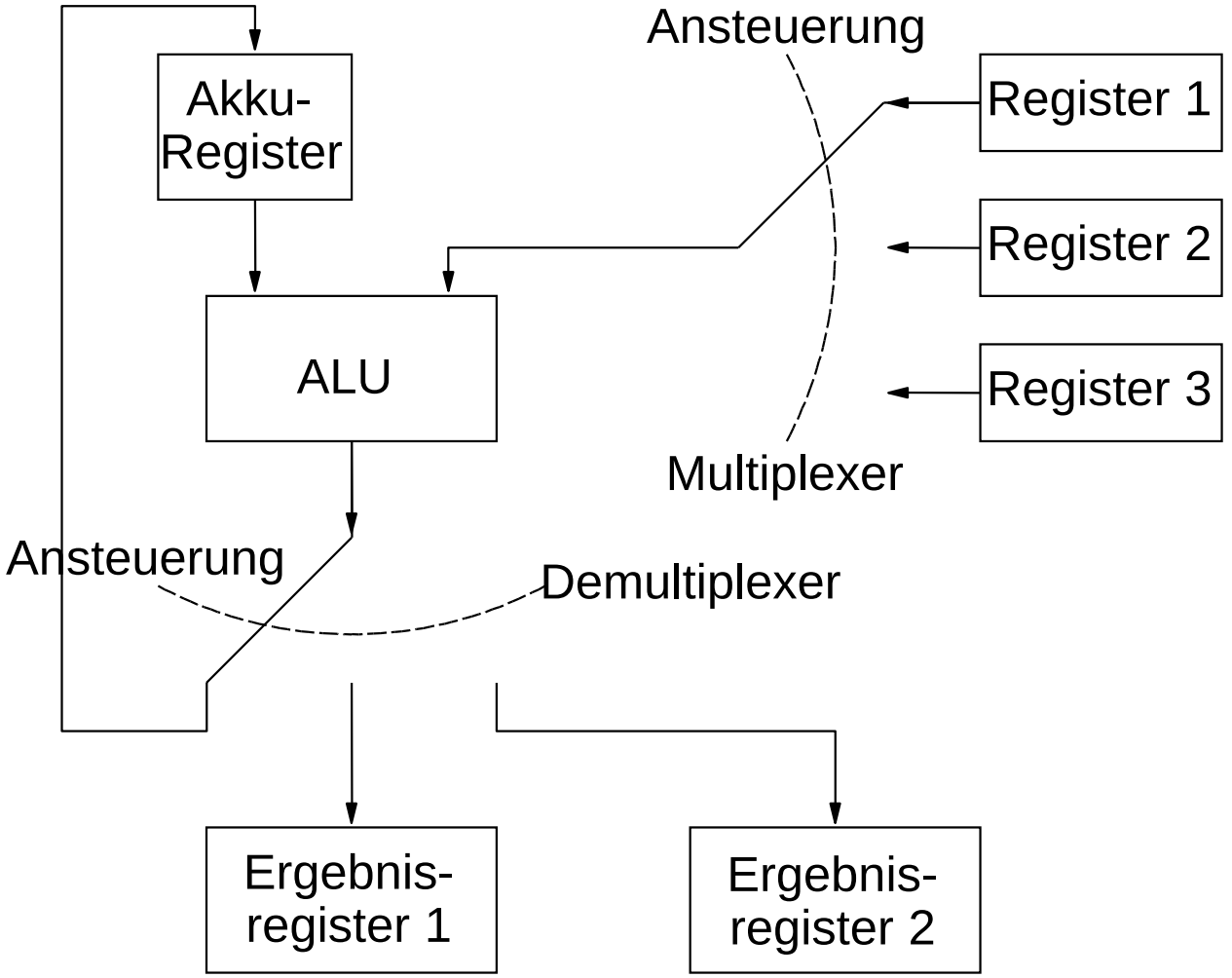
Demultiplexer:



Anwendung – Busbasiert



Anwendung – Multiplexerbasiert

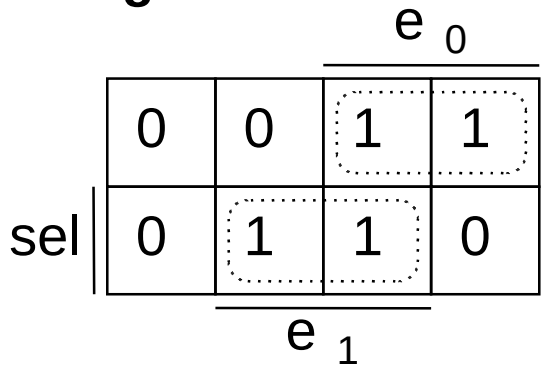


2:1 Multiplexer

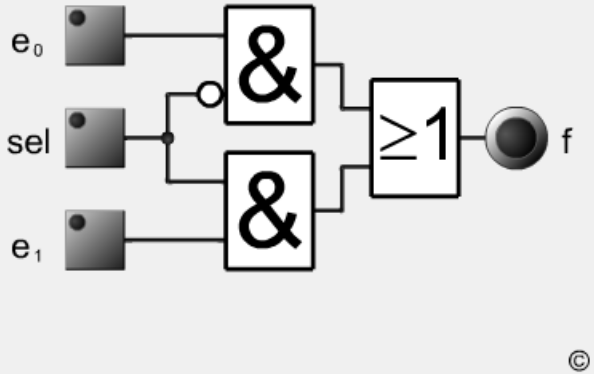
Funktionstafel:

sel	e_0	e_1	f
0	**0**	0	0
0	**0**	1	0
0	**1**	0	1
0	**1**	1	1
1	0	**0**	0
1	0	**1**	1
1	1	**0**	0
1	1	**1**	1

KV-Diagramm:



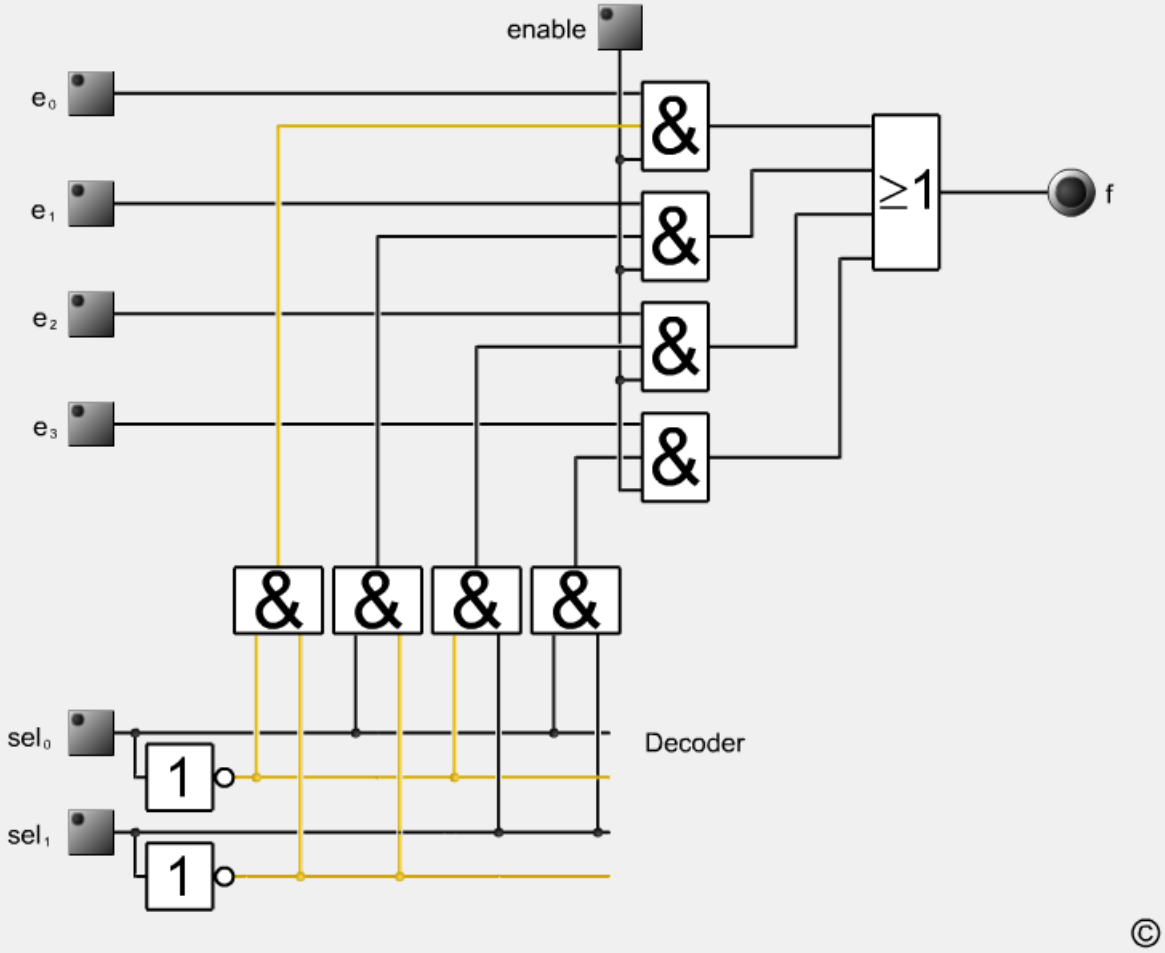
Schaltplan:



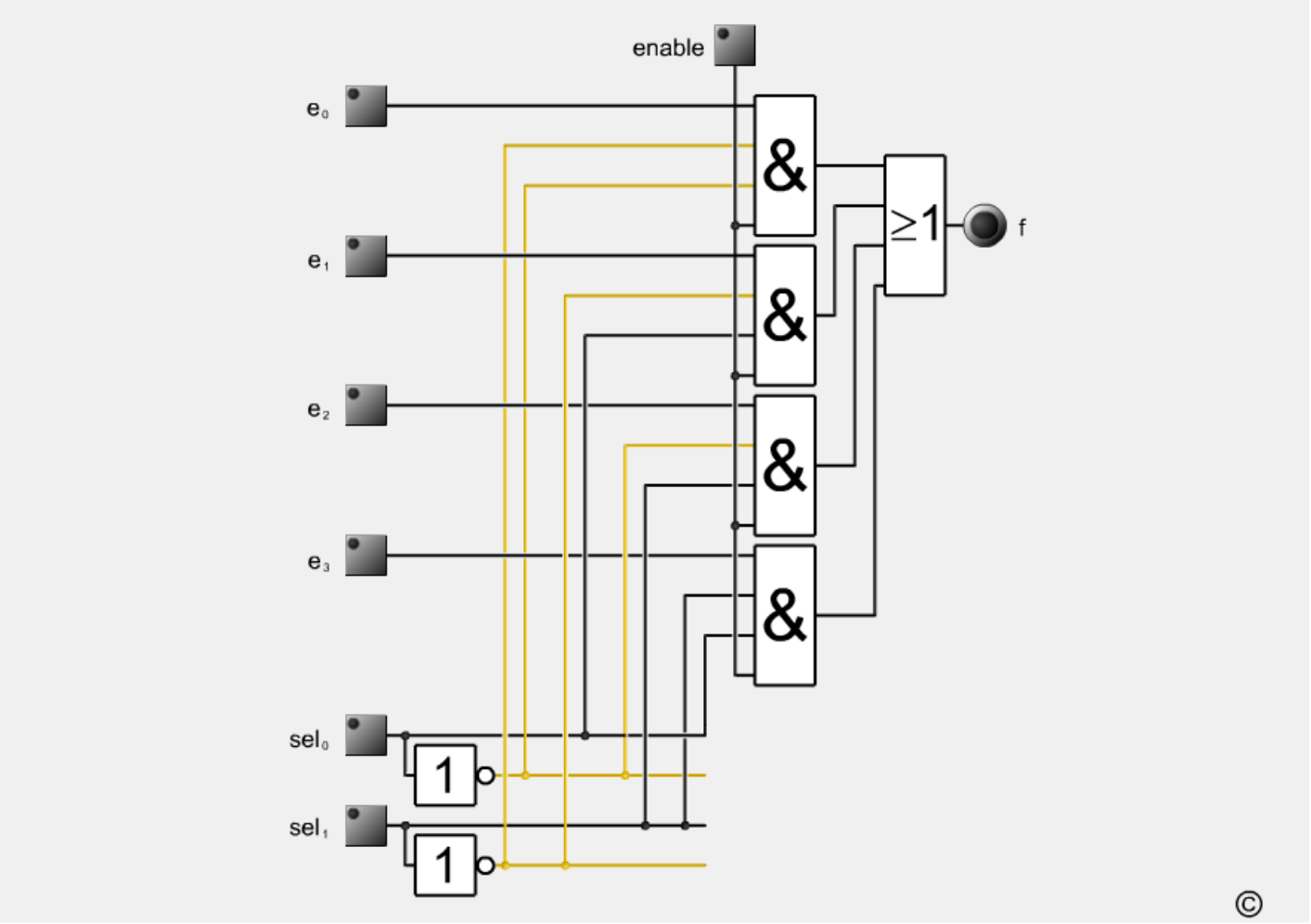
Funktion:

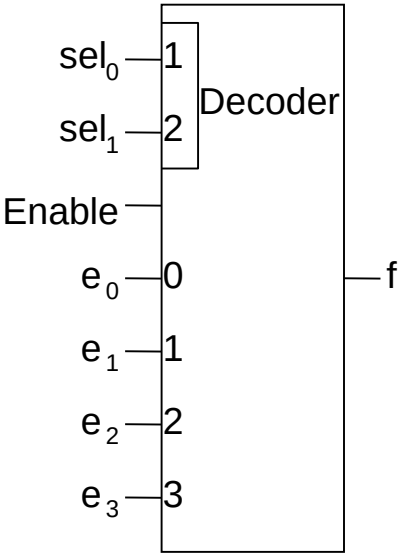
$$f = (\overline{sel} \wedge e_0) \vee (sel \wedge e_1)$$

4:1 Multiplexer

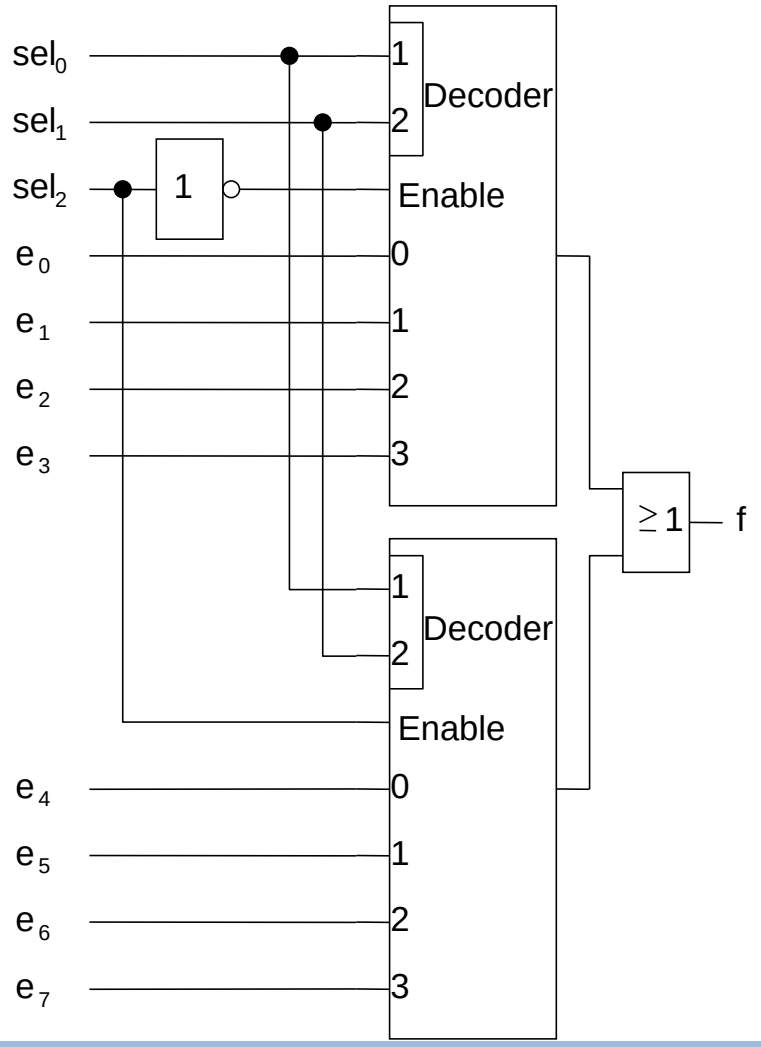


4:1 Multiplexer

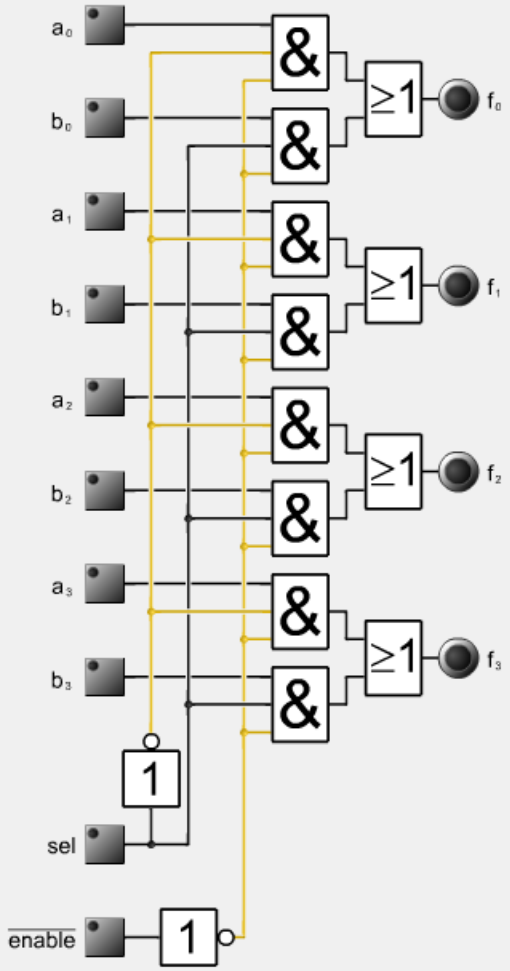




8:1 Multiplexer



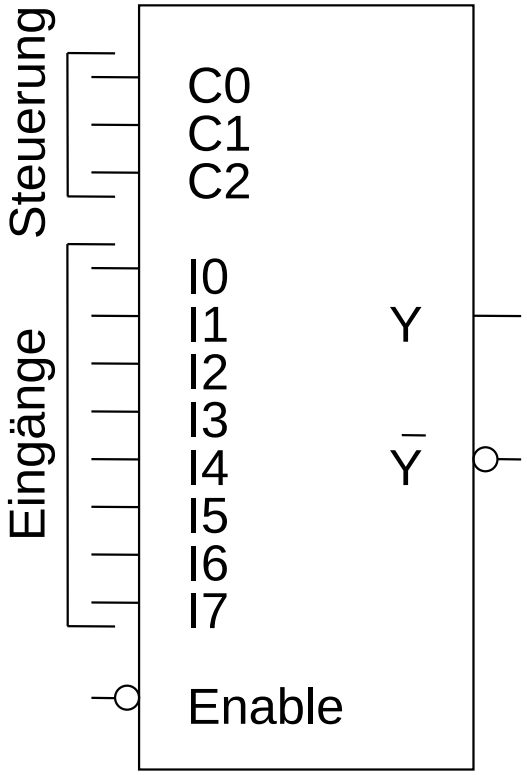
2:1 4-Bit Multiplexer



Multiplexer – Standardbaustein

Multiplexer sind als integrierte Bauelemente erhältlich. Typischerweise handelt es sich dabei um 8:1 bzw. 16:1 Multiplexer (mit 3 bzw. 4 Steuereingängen).

SN74151 (TTL-Baureihe):



Implementierung Boolescher Funktionen

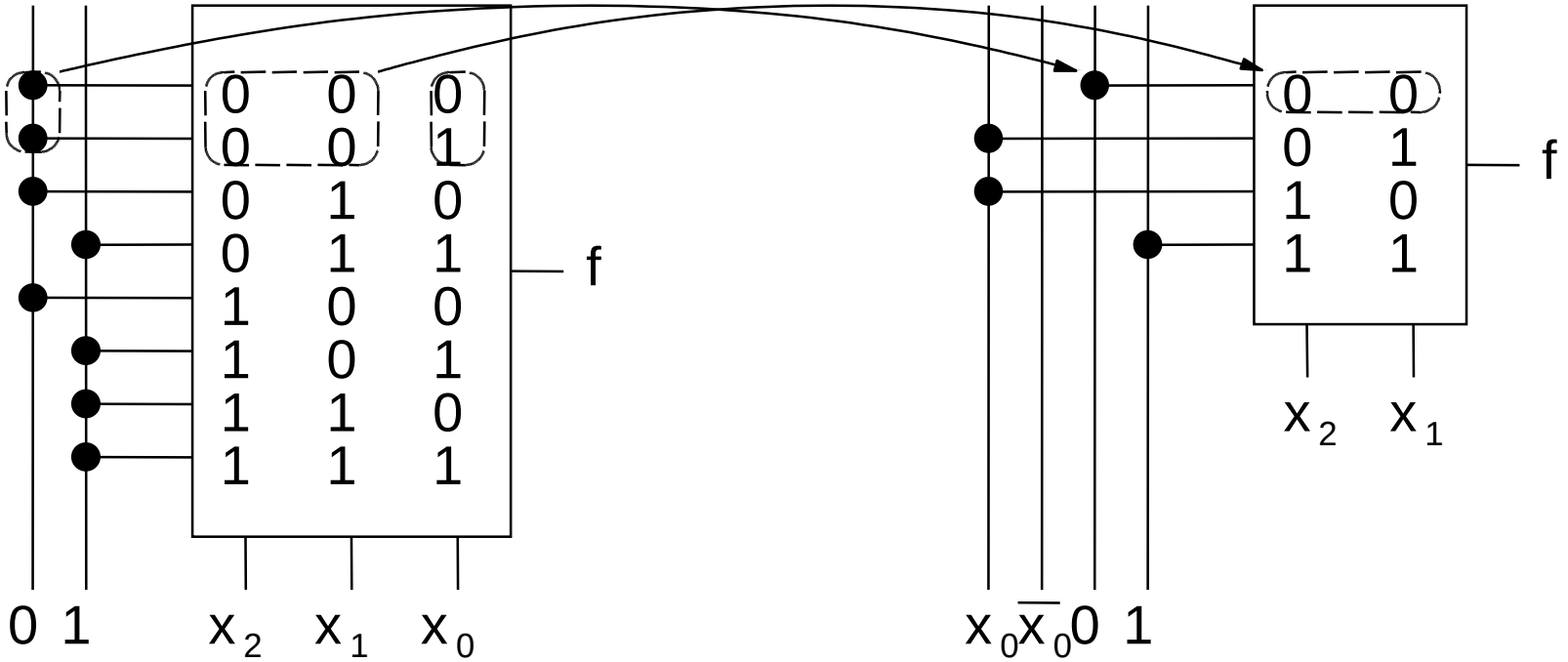
Jede Boolesche Funktion $f(x_0, \ldots, x_{n-1})$ kann unter ausschließlicher Verwendung von Multiplexern realisiert werden. Für f: $\mathbb{B}^n \rightarrow \mathbb{B}$ ist hierzu ein 2^n:1 Multiplexer mit n Steuereingängen notwendig. Die n Eingangsvariablen werden an die Steuereingänge gelegt. Die Belegung der 2^n Dateneingänge erfolgt gemäß der Wahrheitstabelle der Funktion f.

Anstelle eines 2^n:1 Multiplexers kann auch ein 2^{n-1}:1 Multiplexer mit n-1 Steuereingängen verwendet werden. Die Steuereingänge werden mit den Variablen $x_1, \ldots, x_{n-1}$ beschaltet. Die Dateneingänge werden mit den konstanten Werten 0 und 1 und der freien Variablen x_0 bzw. $\overline{x}_0$ belegt.

Implementierung Boolescher Funktionen

Beispiel: 2 von 3 Mehrheitsfunktion

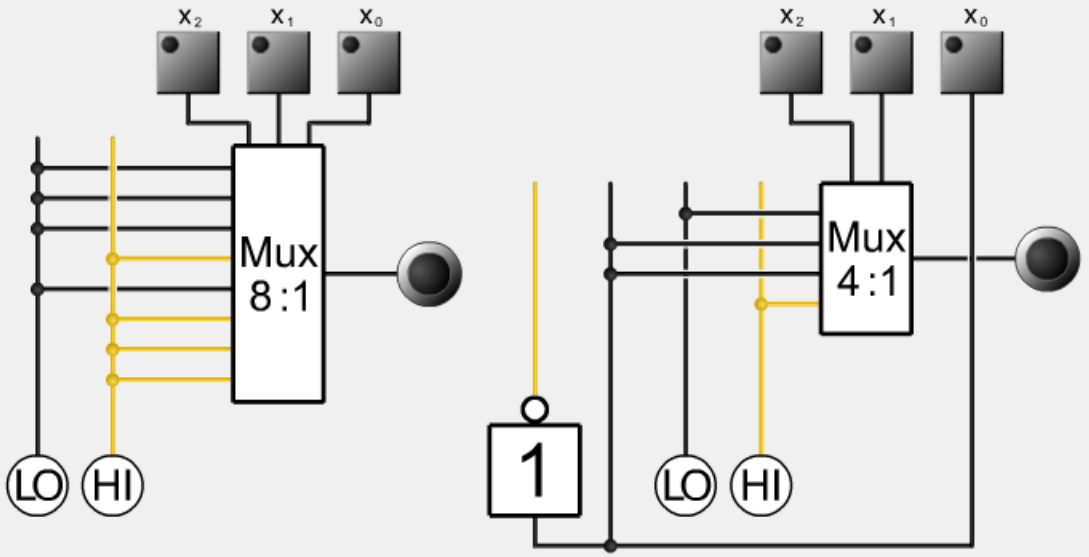
$$f(x_0, x_1, x_2) = x_0x_1 \vee x_0x_2 \vee x_1x_2$$



8:1 Multiplexer

4:1 Multiplexer

Beispiel



Implementierung Boolescher Funktionen

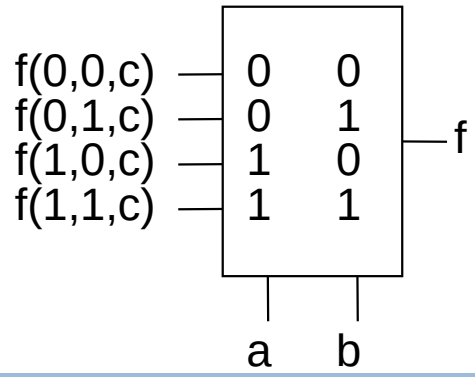
Die Beschaltung der Dateneingänge eines 2^{n-1}:1 Multiplexers erhält man durch n-1-malige Anwendung des Shannon'schen Entwicklungssatzes.

Beispiel: Boolesche Funktion mit 3 Variablen

$$f(a,b,c) = a(\underbrace{f(1,b,c)}_{f(a=1)}) \vee \overline{a}(\underbrace{f(0,b,c)}_{f(a=0)})$$

$$= \underbrace{a(b(\underbrace{f(1,1,c)}_{f(b=1)} \vee \overline{b}(\underbrace{f(1,0,c)}_{f(b=0)})))}_{f(a=1)} \vee \underbrace{\overline{a}(b(\underbrace{f(0,1,c)}_{f(b=1)}) \vee \overline{b}(\underbrace{f(0,0,c)}_{f(b=0)}))}_{f(a=0)}$$

Die Steuereingänge werden mit den für die Entwicklung gewählten Variablen und die Dateneingänge mit den entsprechenden Co-Faktoren belegt.

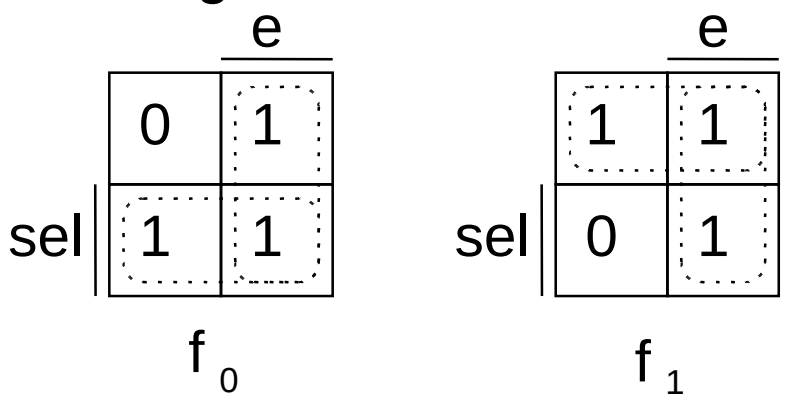


1:2 Demultiplexer

Funktionstafel:

sel	e	f_0	f_1
0	0	**0**	1
0	1	**1**	1
1	0	1	**0**
1	1	1	**1**

KV-Diagramm:



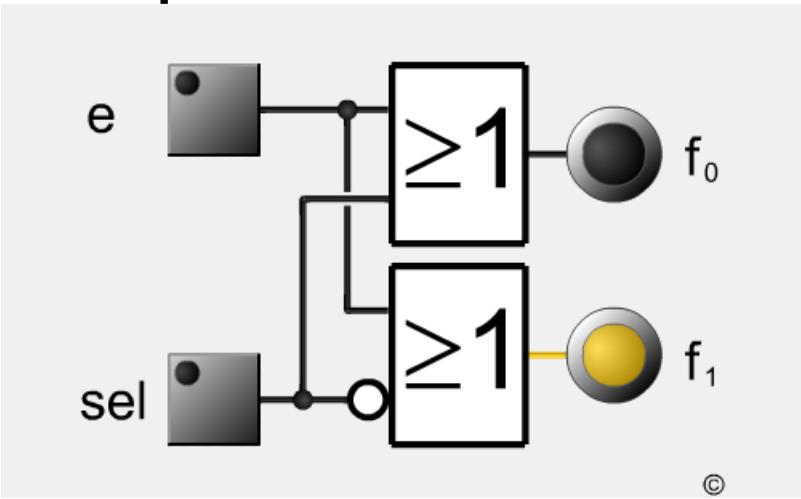
Funktion:

$$f_0 = sel \vee e$$

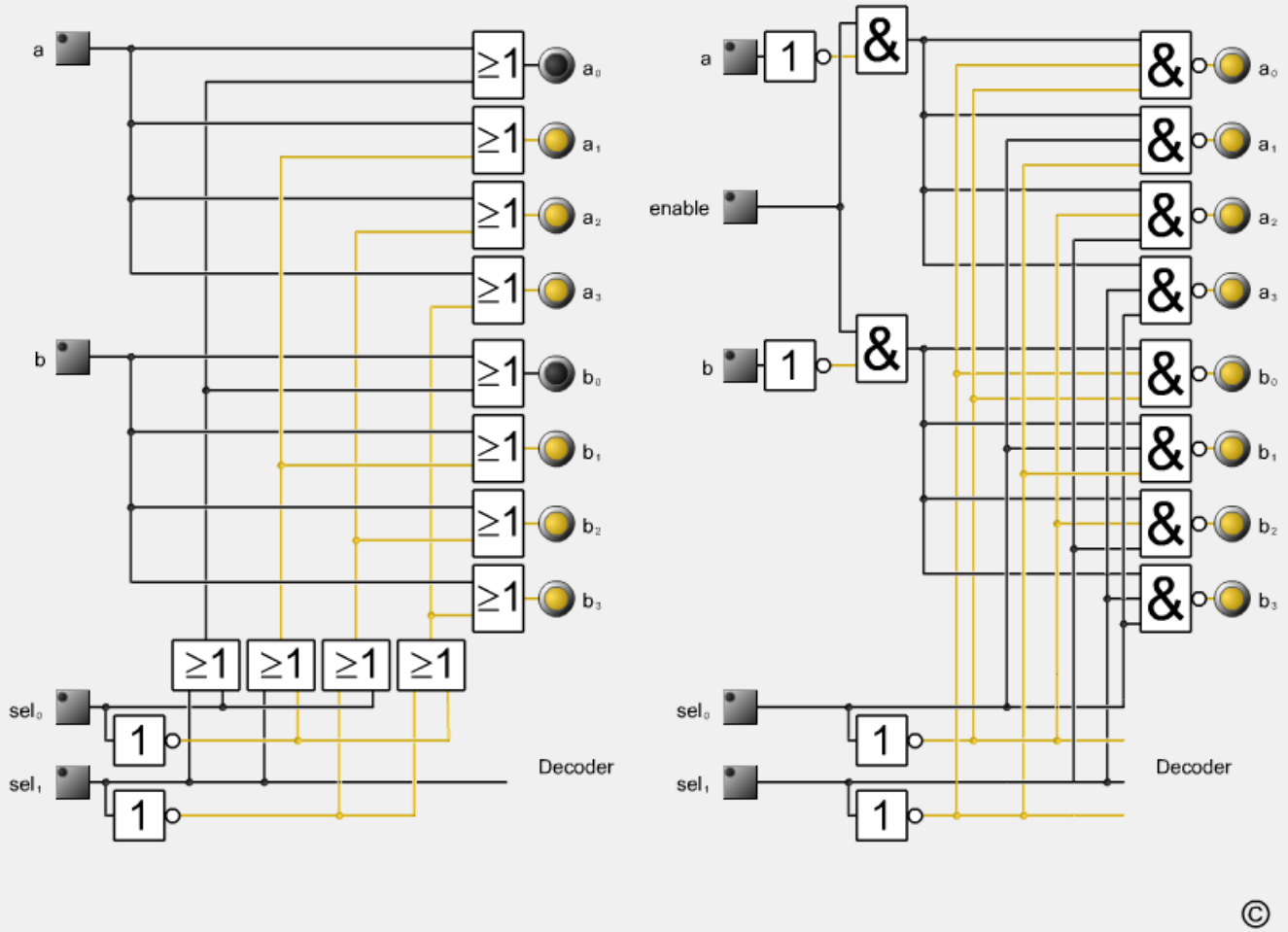
$$f_1 = \overline{sel} \vee e$$

Der nicht beschaltete Ausgang wird mit dem Wert 1 belegt!

Schaltplan:



1:4 2-Bit Demultiplexer



4.1.2 Datenbuszugang

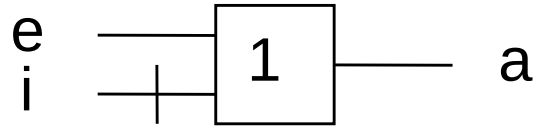
Tri-State-Gatter:

Ein Tri-State-Gatter besitzt die drei definierten Ausgangszustände $0,1$ (abhängig vom Eingangssignal *e*) und einen hochohmigen Zustand z. Der Zustand z wird durch Aktivierung des Sperreingangs i (inhibit) erreicht.

Funktionstafel:

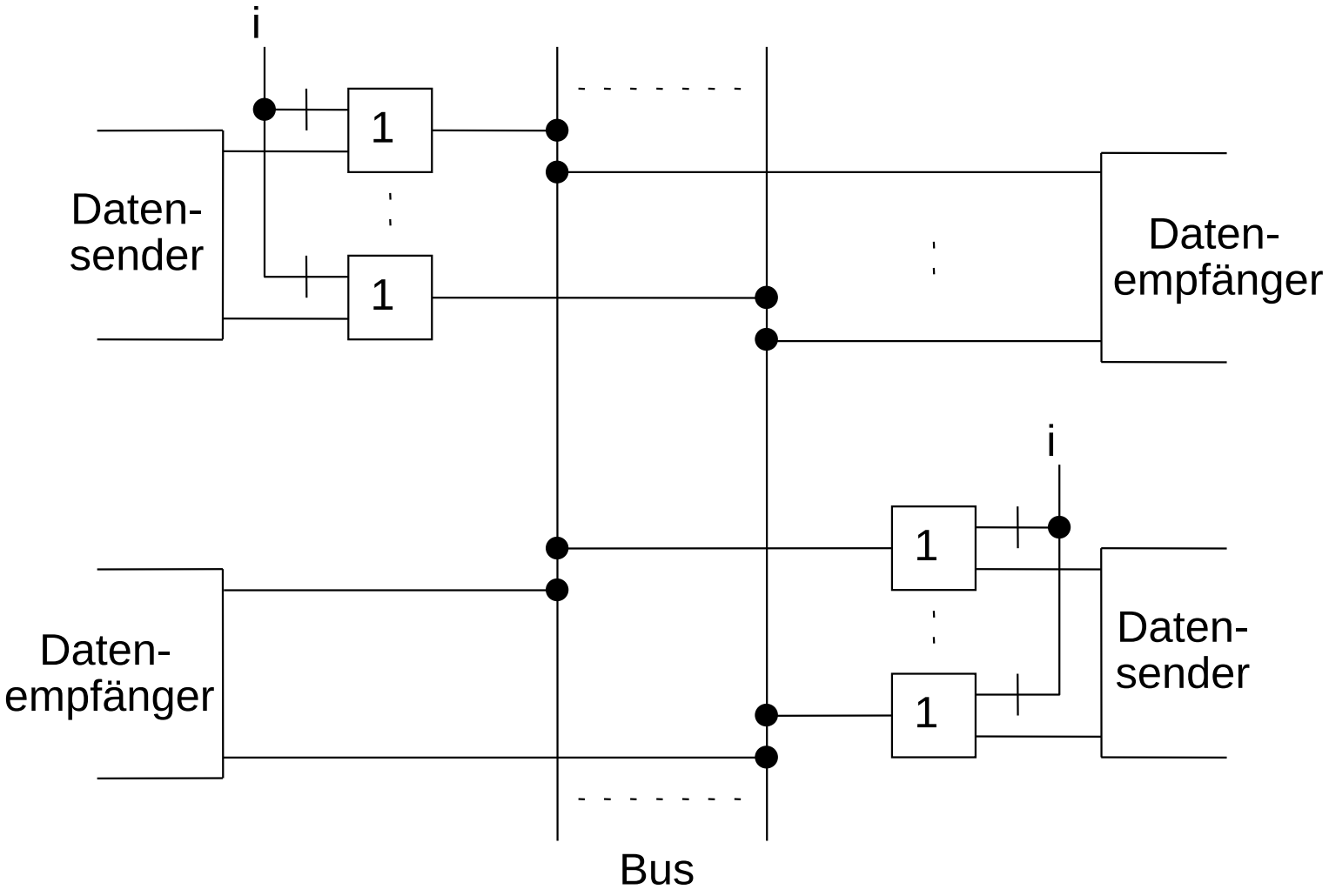
i	e	a
1	–	z
0	0	0
0	1	1

Schaltplan:



Tri-State-Gatter werden immer dann verwendet, wenn *mehrere* Ausgänge, von denen nur einer aktiv sein darf, an eine Leitung angeschlossen werden sollen (z.B. Busleitungen).

Unidirektionaler Datenbuszugang

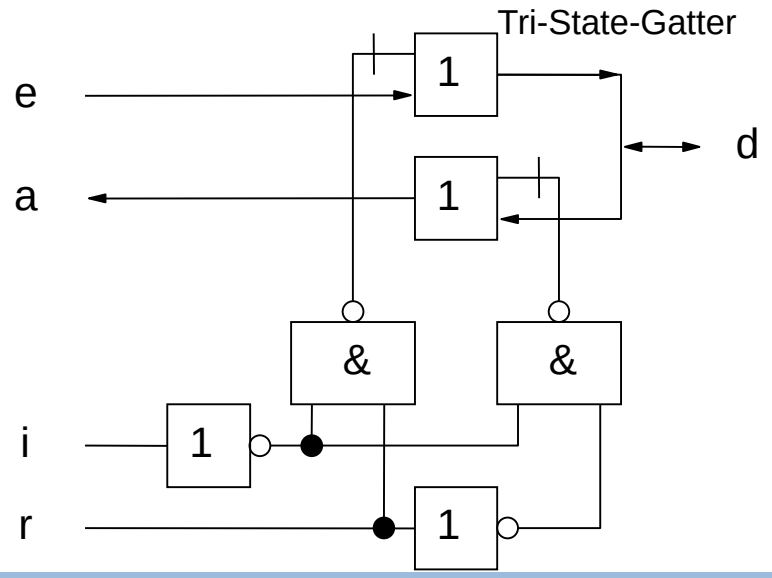


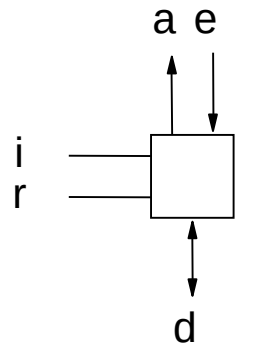
Bidirektionaler Datenbuszugang

Funktionstafel:

Sperreingang i	Richtungssteuerung r	Übertragungsweg
0	0	$d \rightarrow a$ (empfangen)
0	1	$e \rightarrow d$ (senden)
1	0	offen
1	1	offen

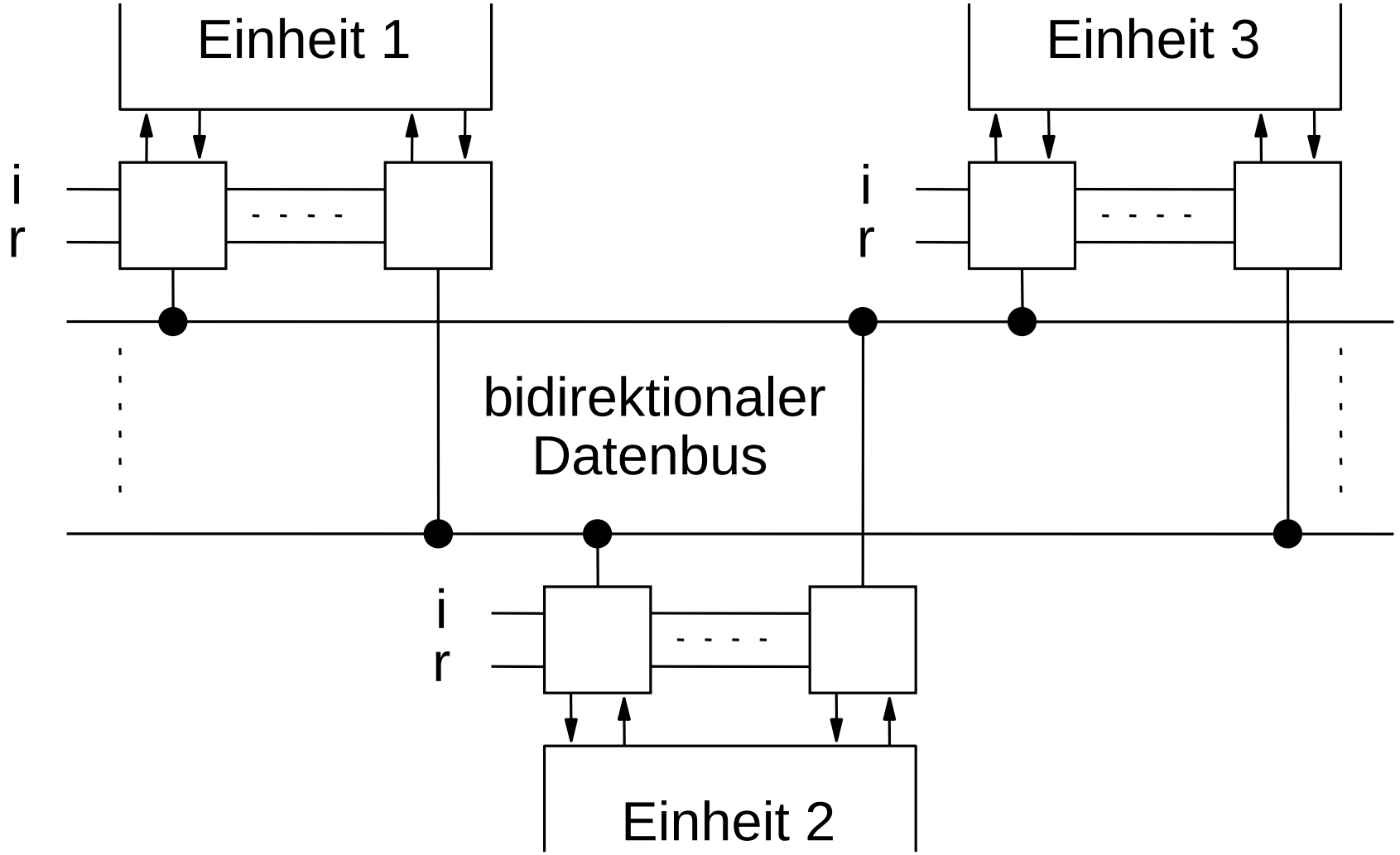
Schaltplan:





Schaltsymbol

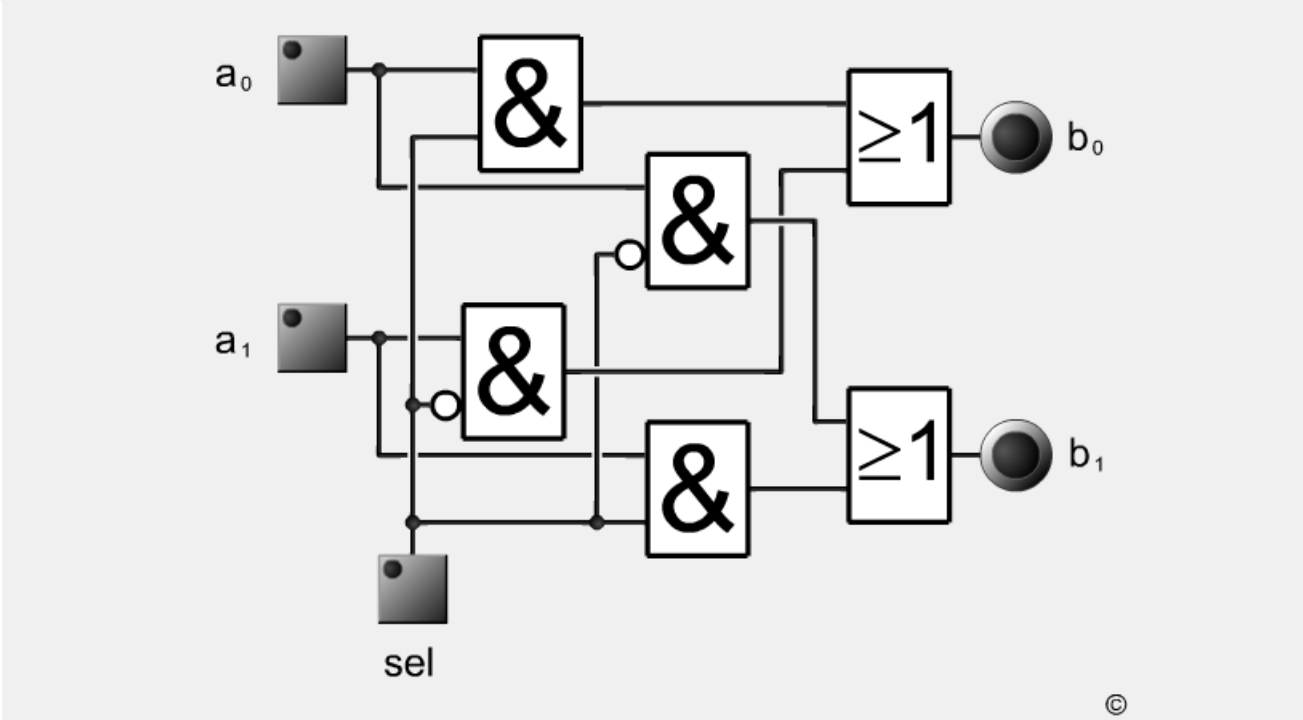
Bidirektionaler Datenbuszugang



4.1.3 Permutationsschaltnetz

Permutationsschaltnetze vertauschen die Reihenfolge von Variablen. Die Vertauschung wird über einen Steuereingang *sel* aktiviert.

Schaltplan:



Permutationsschaltnetze werden in Verbindungsnetzwerken verwendet, um verschiedene Kommunikationswege schalten zu können.

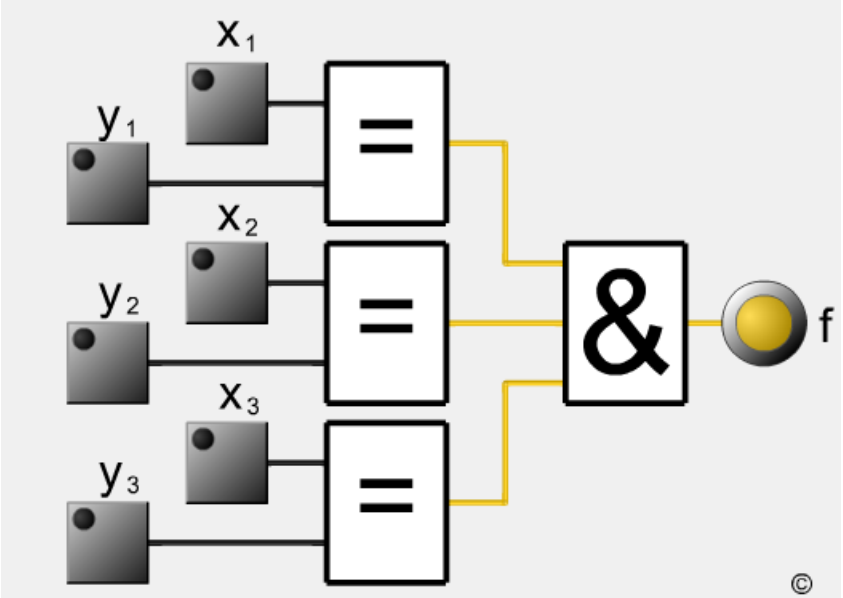
4.1.4 Vergleicher (Komparator)

Vergleicher sind Schaltungen, die insbesondere in Mikroprozessoren Verwendung finden um beispielsweise den notwendigen Vergleich für bedingte Sprunganweisungen durchzuführen.

Vergleicher werden aber auch integriert in Schaltungen eingesetzt, z.B. zur Speicherauswahl oder für die Selektion von Ein-/Ausgabe-Geräten.

Vergleich der 2 Booleschen Tupel X und Y:

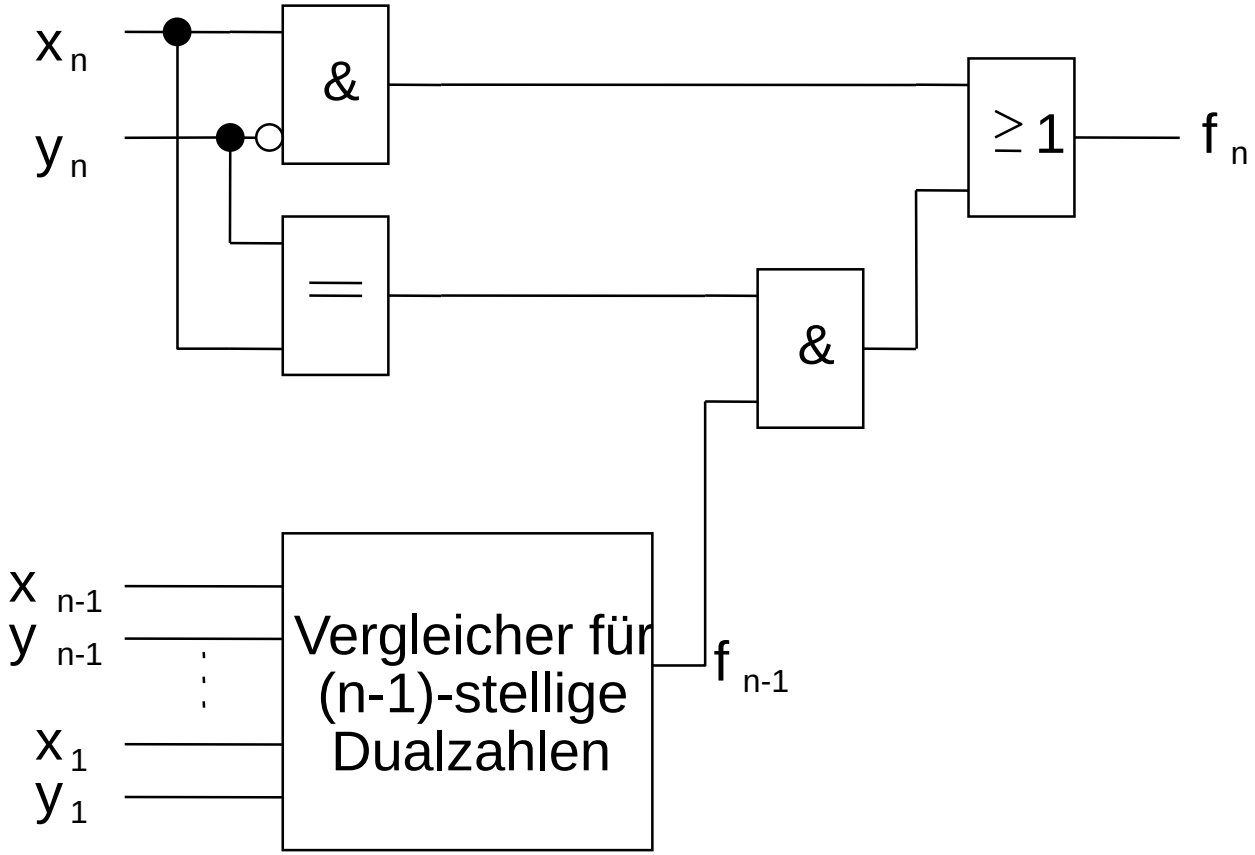
$$f(X,Y) = \bigwedge_{i=1}^{n} (x_i \Leftrightarrow y_i)$$



n-Bit Vergleicher

Vergleicher für n-Bit Zahlen lassen sich hierarchisch konstruieren.

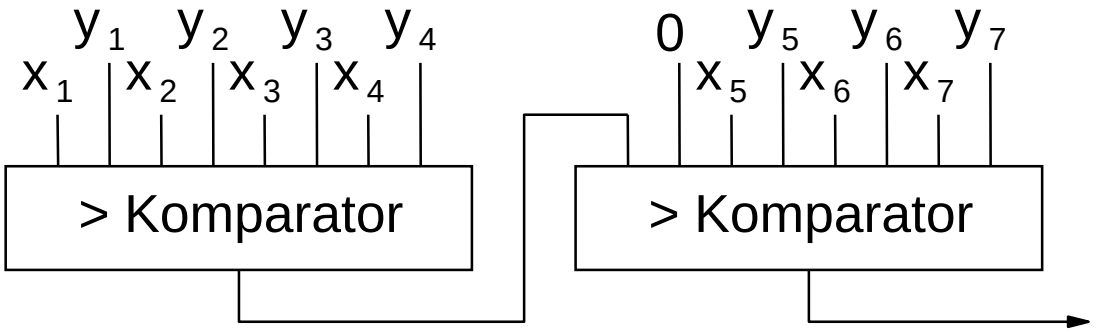
($x \geq y$)-Vergleicher:



n-Bit Vergleicher

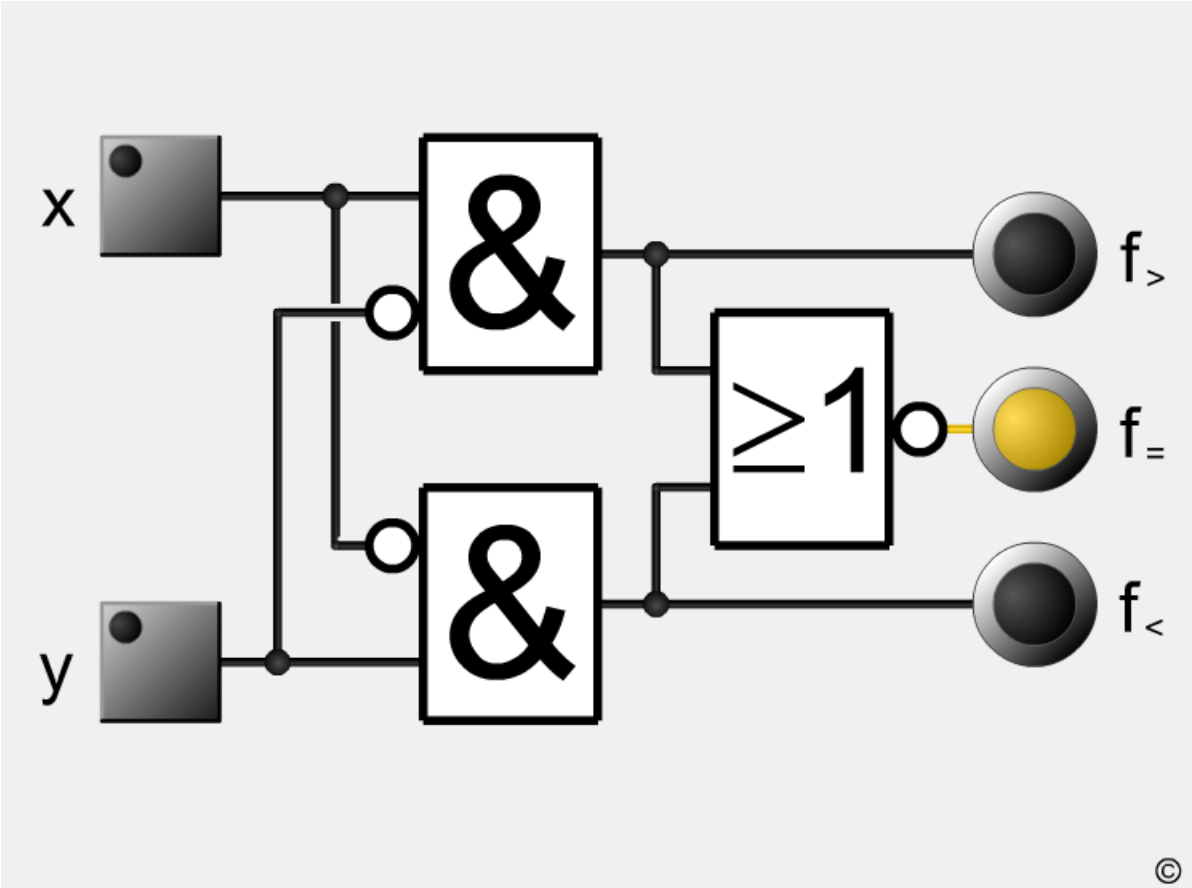
n-Bit Vergleicher lassen sich zu Vergleichern von Dualzahlen mit größeren Längen kaskadieren.

7-Bit Vergleicher aus zwei 4-Bit Vergleichern:



1-Bit Vergleicher für Dualzahlen

Der größer als bzw. kleiner als Vergleicher ist schaltungstechnisch aufwendiger als der Vergleich auf Identität.



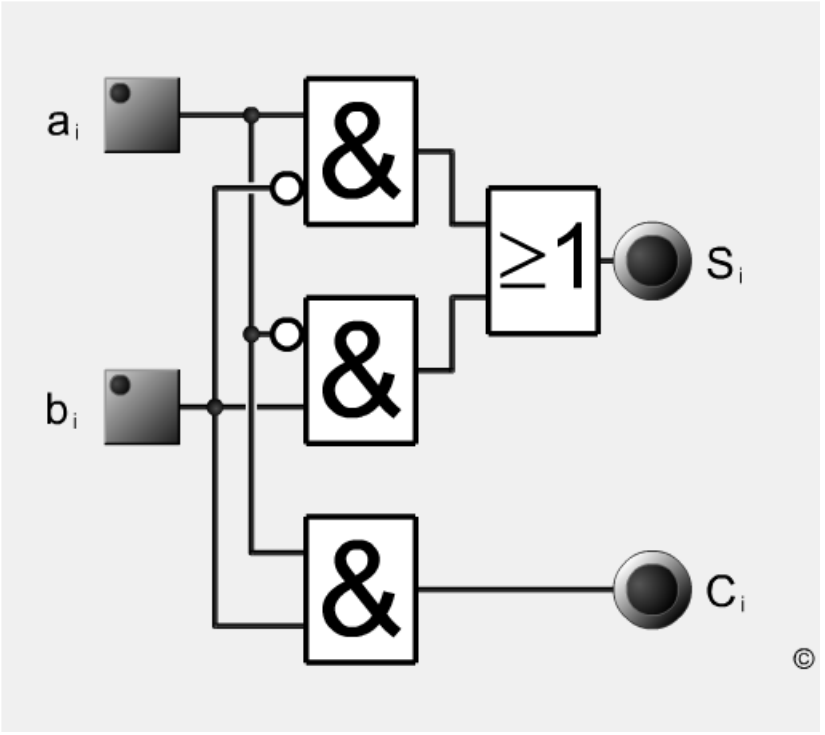
4.1.5 Addierer

Ein **Halbaddierer** berechnet aus zwei 1-Bit Zahlen die Summen S_i und den Übertrag C_{i+1}.

Funktionstafel:

a_i	b_i	S_i	C_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Schaltplan:



Funktionen:

$$S_i = a_i \oplus b_i = \overline{a_i} b_i \vee a_i \overline{b_i}$$

$$C_{i+1} = a_i b_i$$

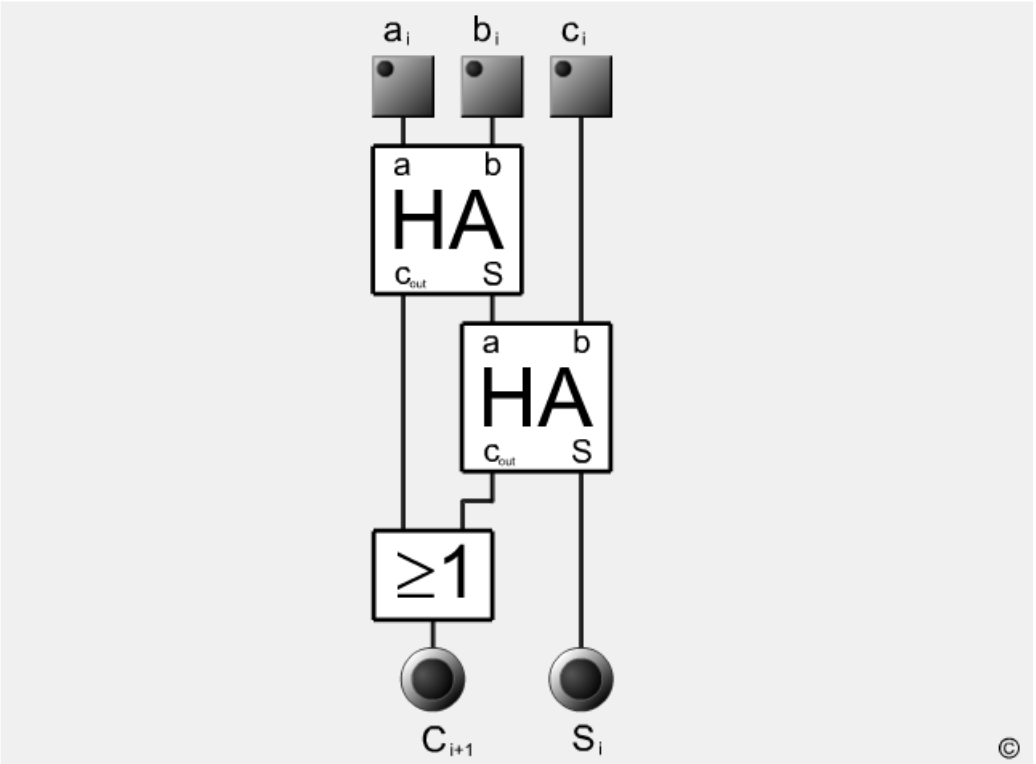
Volladdierer

Ein **Volladdierer** berechnet aus drei 1-Bit Zahlen die Summe S_i und den Übertrag C_{i+1}.

Funktionstafel:

a_i	b_i	C_i	S_i	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

Schaltplan:



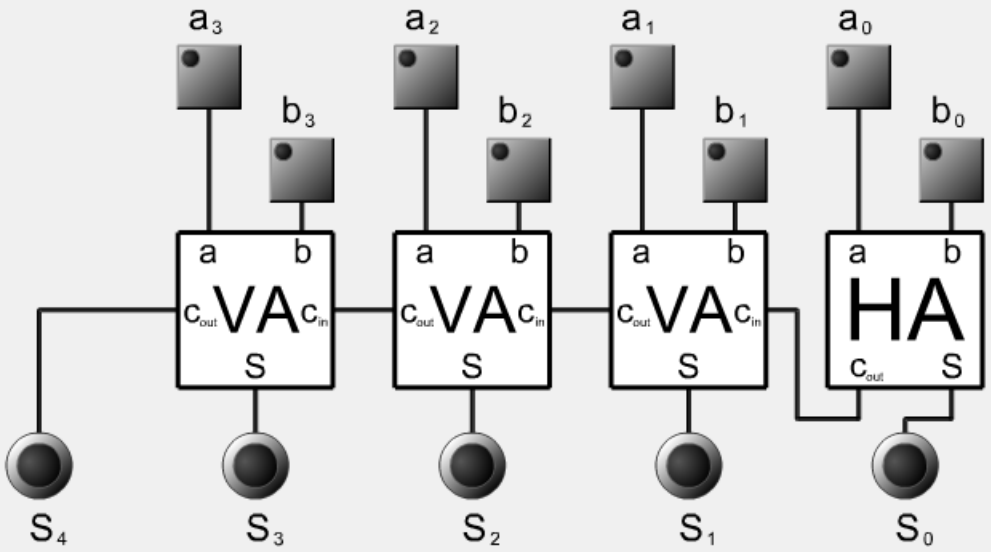
Funktionen:

$$S_i = a_i \oplus b_i \oplus C_i$$

$$C_{i+1} = a_i b_i \vee a_i C_i \vee b_i C_i$$

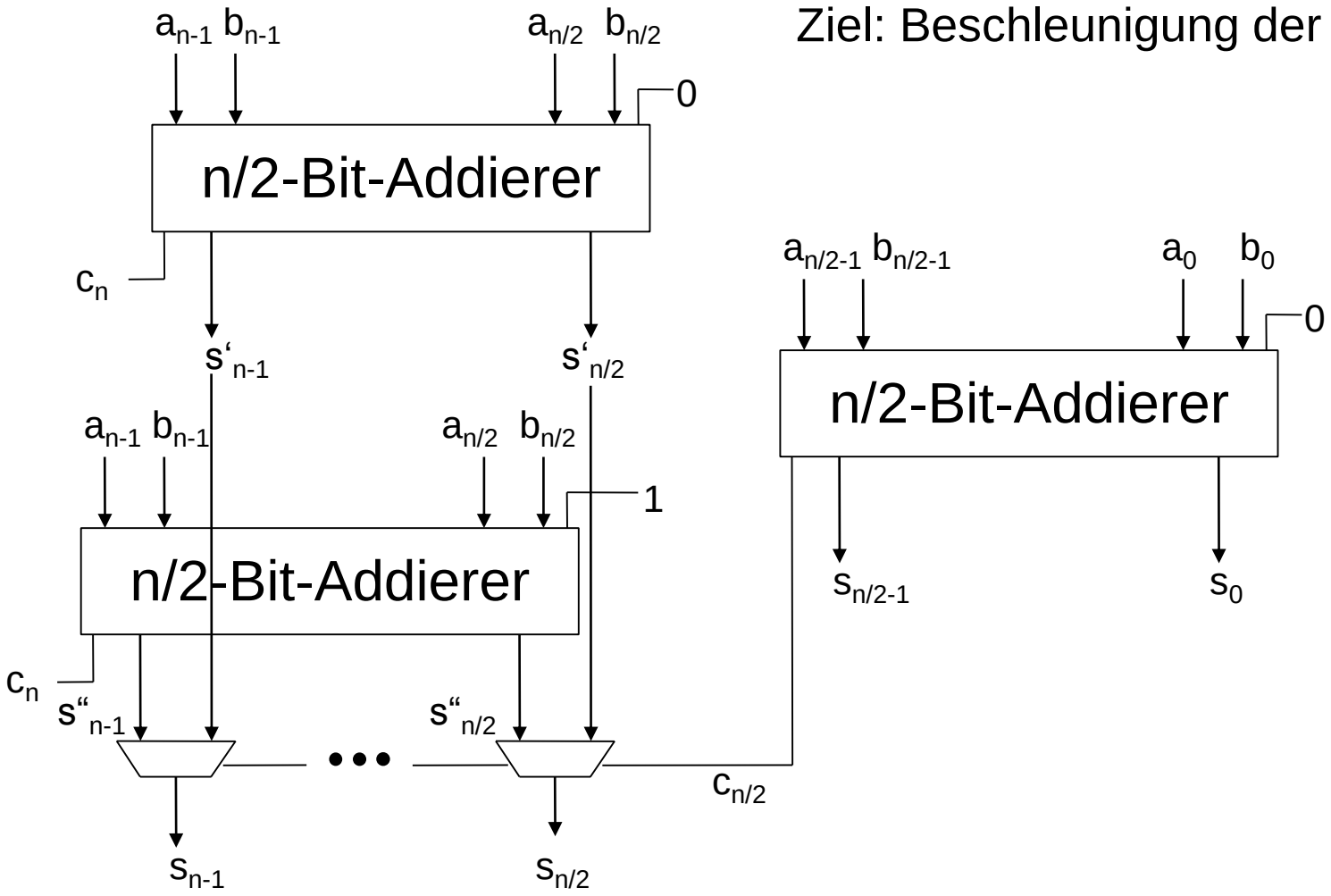
n-Bit Ripple-Carry-Addierer

Ein n-Bit Ripple-Carry-Addierer entsteht durch die Kaskadierung von n-1 Volladdierern und einem Halbaddierer.



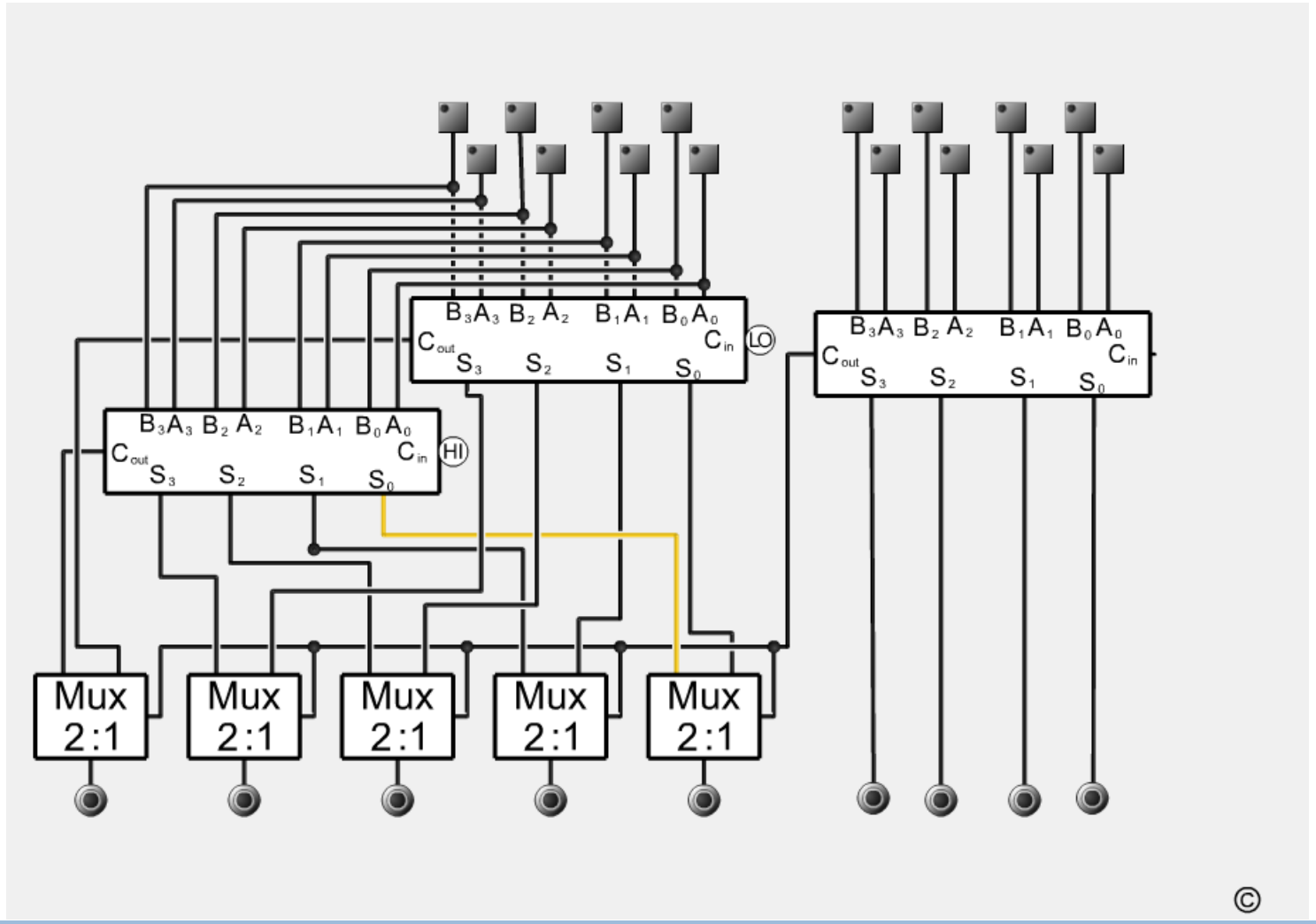
Im schlimmsten Fall müssen für die Addition zweier n-Bit Zahlen alle Addierer nacheinander durchlaufen werden, bis der Übertrag S_n vorliegt.

Carry-Select-Addierer



Ziel: Beschleunigung der Durchlaufzeit

Carry Select Addierer



Subtraktionsschaltung aus Volladdierern

Die Differenz A-B ($A,B \geq 0$) wird durch die Addition des 2-er Komplements $\neg B$ von B erreicht ($A+(\neg B)$), welches auf die bitweise Negation $\overline{B}$ von B zurückgeführt wird:

$$B + \neg B = 2^n \quad \Rightarrow \quad \neg B = 2^n - B = \overline{B} + 1 \quad \text{da} \quad B + \overline{B} = 2^n - 1$$

$$S = A - B = A + (2^n - B) = A + \overline{B} + 1$$

Auftretende Überträge werden nicht berücksichtigt.

Beispiel: 4-Bit (Vorzeichen und 3-Bit für die Zahlendarstellung)

	B	$\overline{B}$	
0	0 000	1 111	-0
1	0 001	1 110	-1
2	0 010	1 101	-2
⋮	⋮	⋮	⋮
7	0 111	1 000	-7

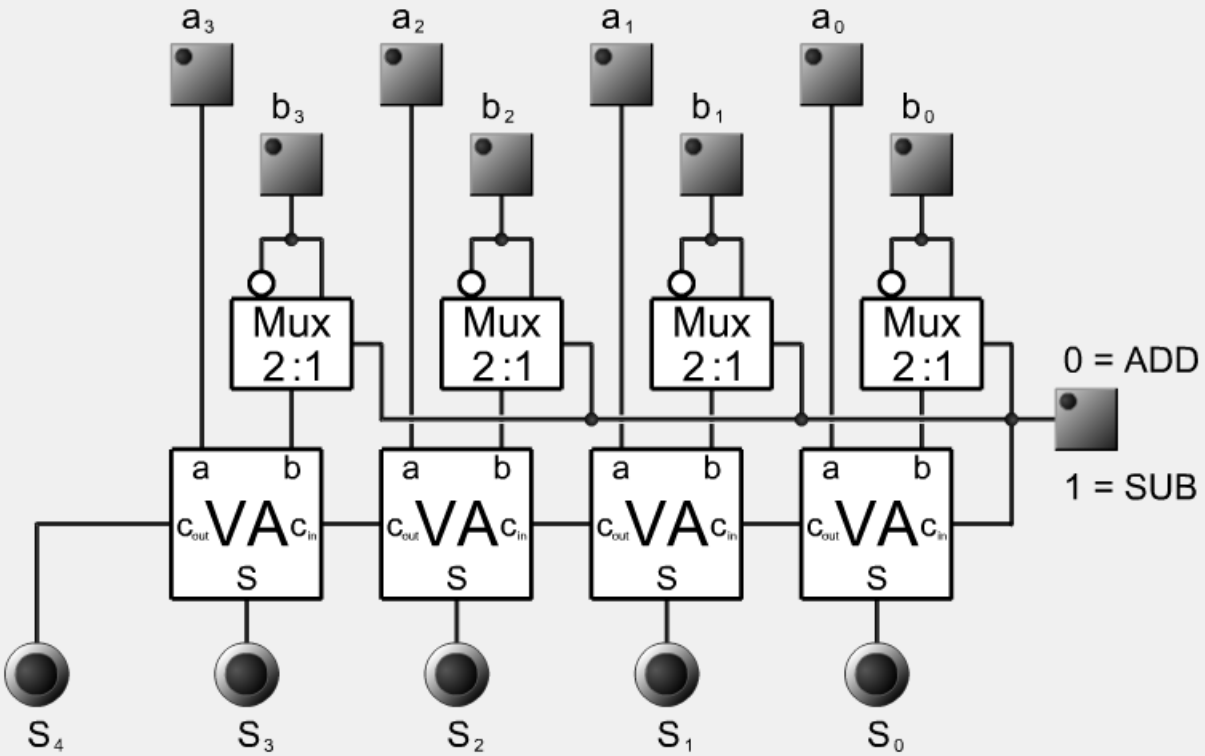
1-er Komplement

	B	$\neg B$	
0	0 000	1 111	-1
1	0 001	1 110	-2
2	0 010	1 101	-3
⋮	⋮	⋮	⋮
7	0 111	1 000	-8

2-er Komplement

Subtraktionsschaltung auf Volladdierern

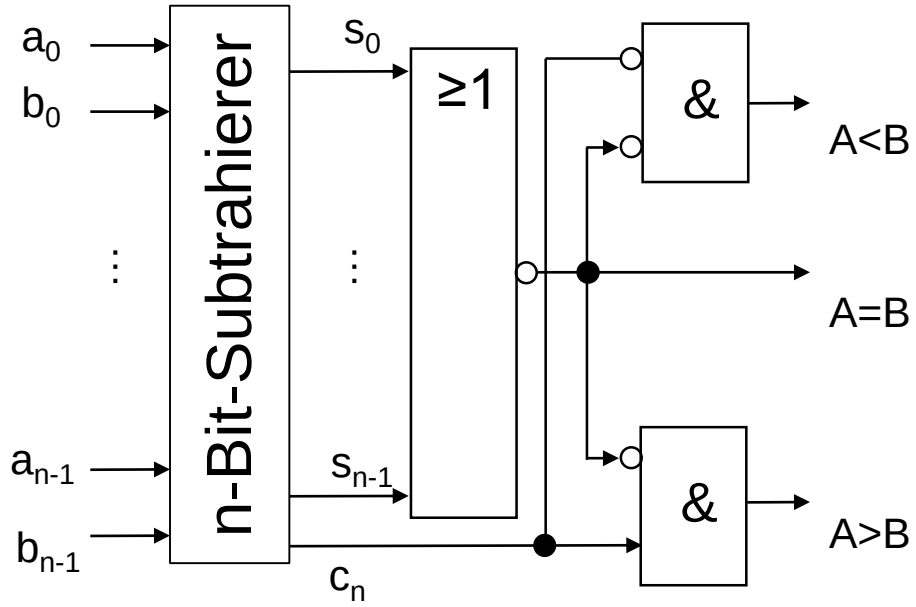
Addierer/Subtrahierer:

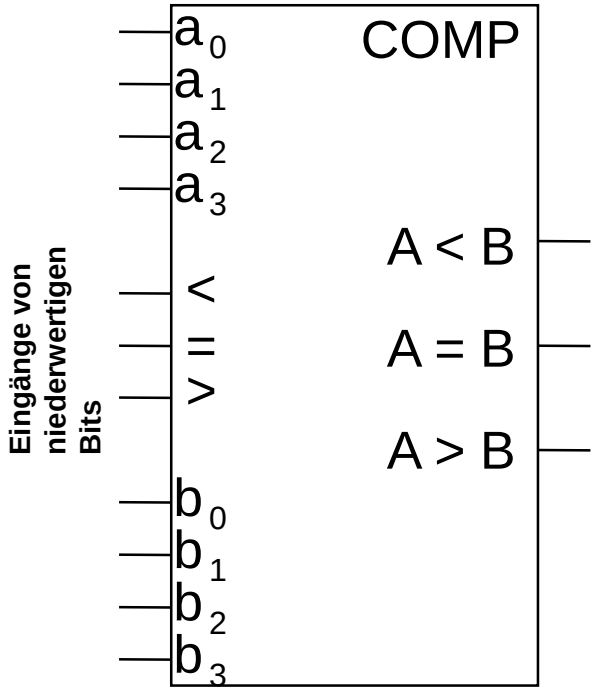


Allgemeine Vergleicher für positive Binärzahlen

Der größer als bzw. kleiner als Vergleicher basiert auf einem Subtrahierer.

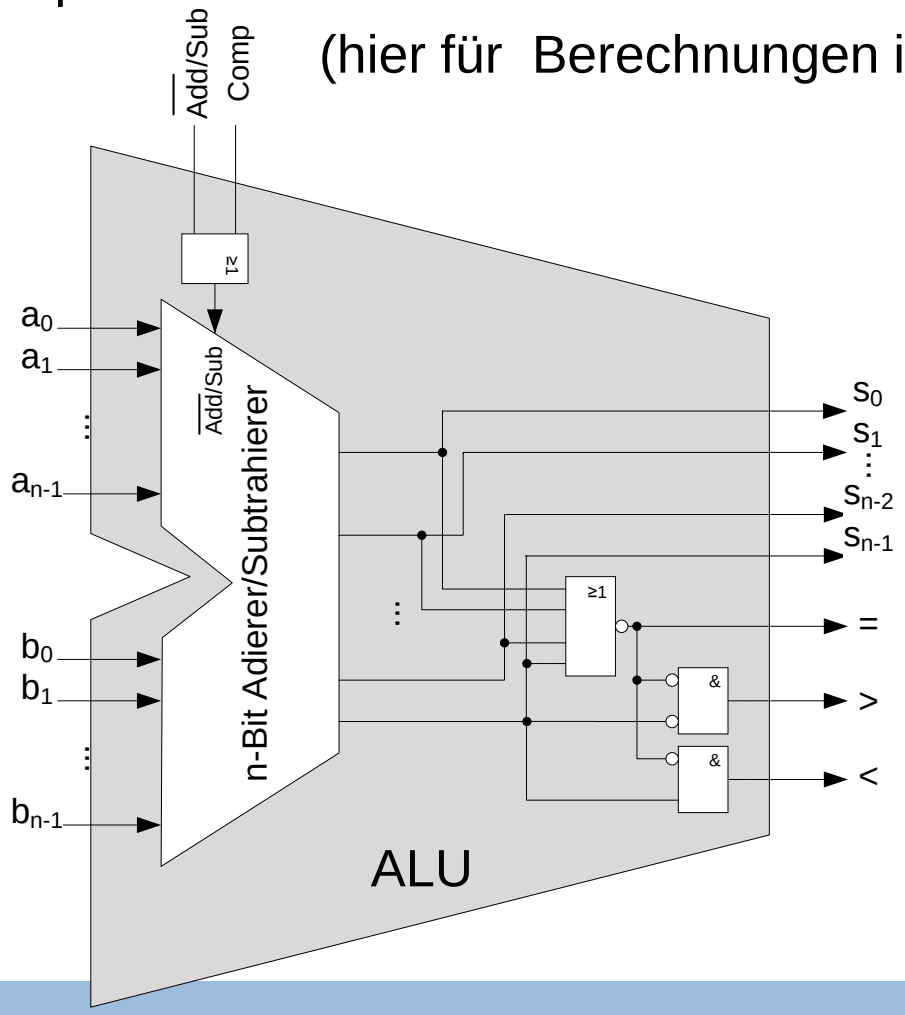
Integrierter Vergleicher (Komparator) SN7485:





ALU

Kombiniert man Komparator und Addierer/Subtrahierer erhält man eine ALU (hier für Berechnungen im 2er Komplement)



4.1.6 Multiplizierer

Schnelle Multiplizierer werden durch parallele Berechnung sämtlicher (dualer) Produktterme und anschließender Addition der Terme mit den richtigen Wertigkeiten implementiert.

$$a = a_0 * 2^0 + a_1 * 2^1 + \ldots + a_{m-1} * 2^{m-1}$$
$$b = b_0 * 2^0 + b_1 * 2^1 + \ldots + b_{m-1} * 2^{m-1}$$

$$p = a * b$$

$$\begin{aligned} p = {} & (a_0 b_0) * 2^0 \\ & + (a_0 b_1 + a_1 b_0) * 2^1 \\ & + (a_0 b_2 + a_1 b_1 + a_2 b_0) * 2^2 \\ & + \ldots \\ & + \left(\sum_{i=0}^{k} a_i b_{k-i} \right) * 2^k \\ & + \ldots \\ & + (a_{m-1} b_{m-1}) * 2^{2m-2} \end{aligned}$$

Es werden m^2 viele AND-Gatter für die Bildung der Produktterme und m Addierer benötigt.

Beispiel – 3-Bit Multiplizierer (1)

Eingaben: Faktoren a und b

$$a = a_0 * 2^0 + a_1 * 2^1 + a_2 * 2^2$$
$$b = b_0 * 2^0 + b_1 * 2^1 + b_2 * 2^2$$

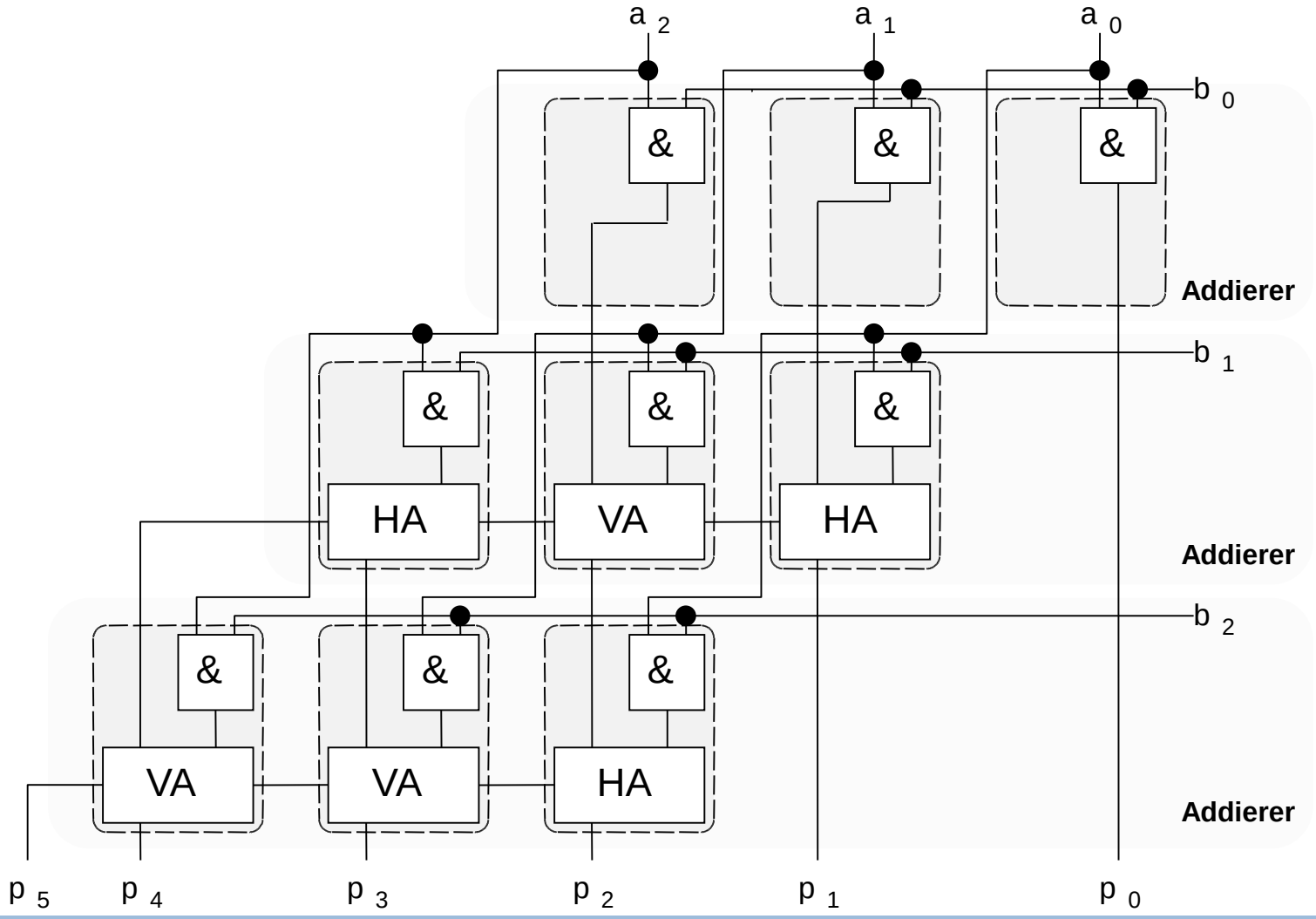
Ausgabe: Produkt $p = a \cdot b$

$$\begin{aligned} p = {} & (a_0 b_0) * 2^0 \\ & + (a_1 b_0 + a_0 b_1) * 2^1 \\ & + (a_2 b_0 + a_1 b_1 + a_0 b_2) * 2^2 \\ & + (a_2 b_1 + a_1 b_2) * 2^3 \\ & + (a_2 b_2) * 2^4 \end{aligned}$$

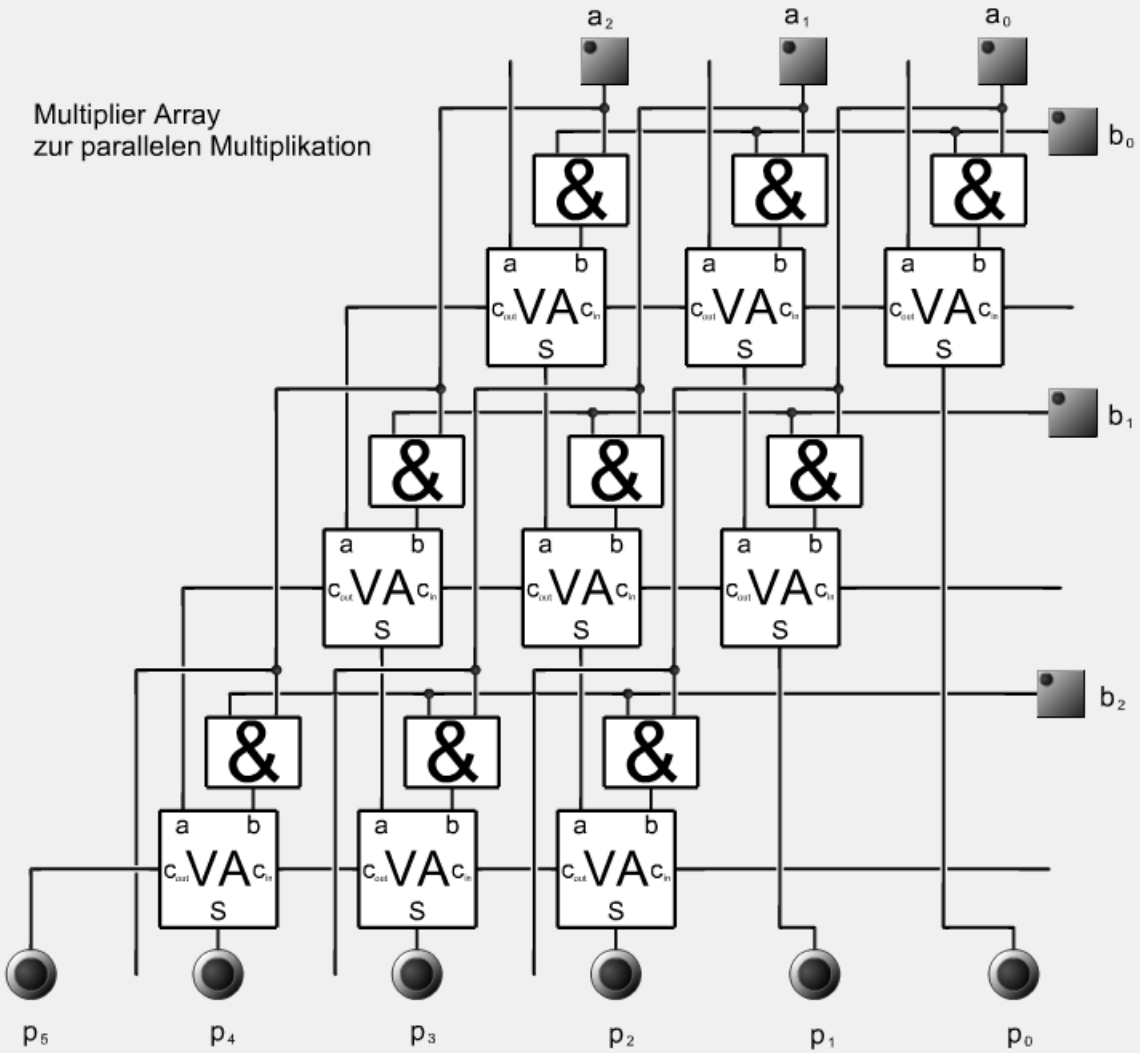
Anschauliche Darstellung:

a_2	a_1	a_0	$*$	b_2	b_1	b_0
				$a_2 b_0$	$a_1 b_0$	$a_0 b_0$
			$a_2 b_1$	$a_1 b_1$	$a_0 b_1$	
		$a_2 b_2$	$a_1 b_2$	$a_0 b_2$		
p_5		p_4	p_3	p_2	p_1	p_0

Beispiel – 3-Bit Multiplizierer (1)

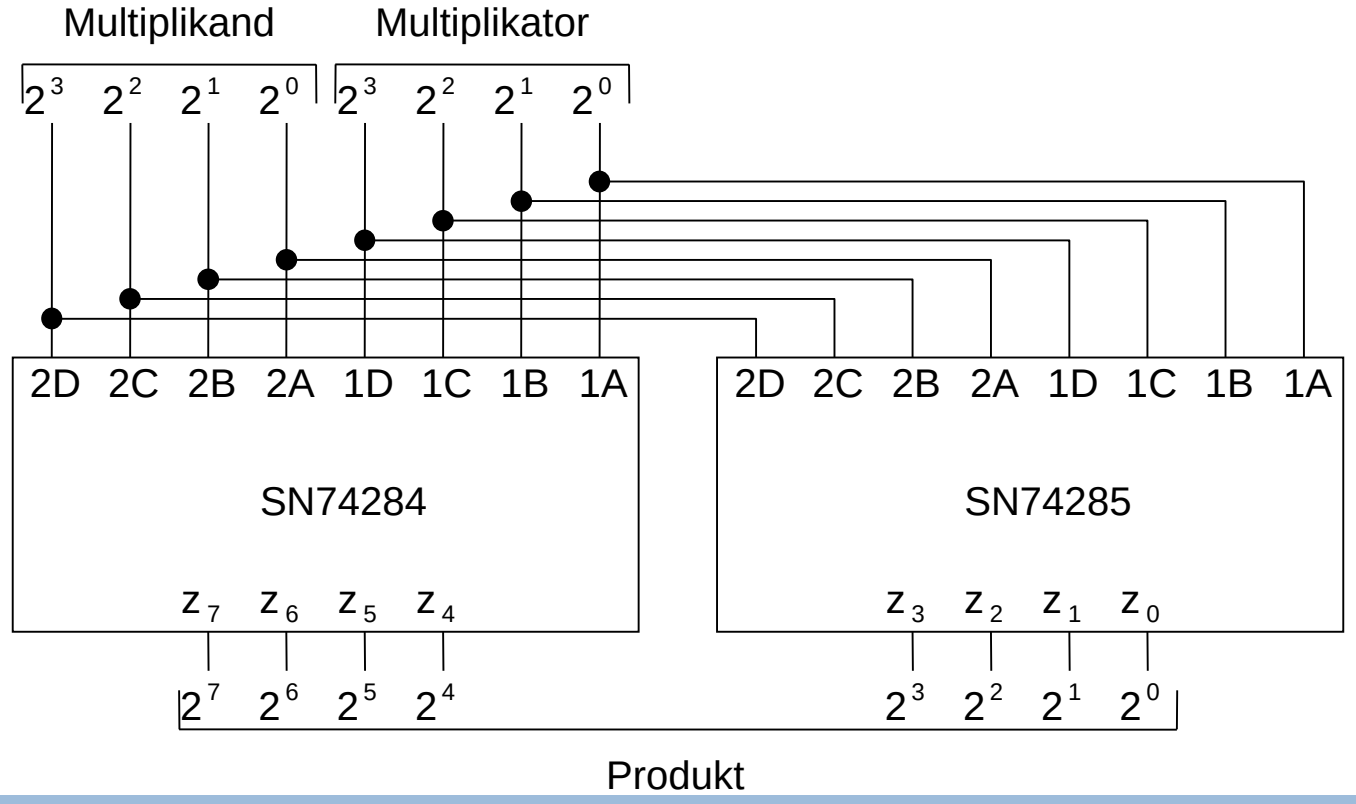


Beispiel – 3-Bit Multiplizierer (2)



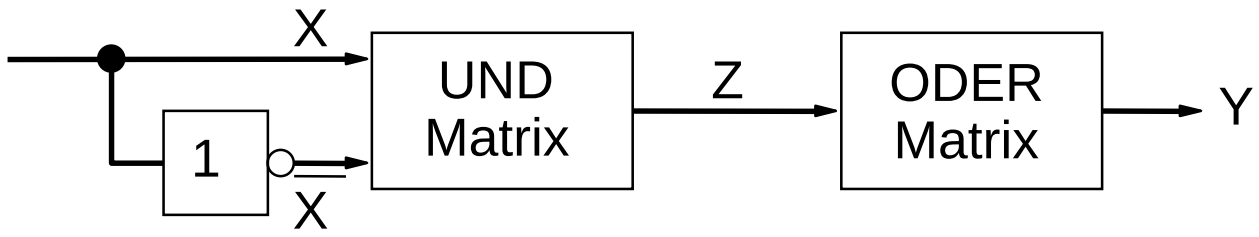
Standardbausteine für Multiplikation

Für die Multiplikation zweier 4-Bit Dualzahlen existieren die beiden Standardbausteine SN74284 und SN74285. Der Baustein SN74285 berechnet den niederwertigen 4-stelligen Teil des Produktes, der SN74284 den höherwertigen Teil.



4.2 PLA (programmable logic arrays)

PLAs sind integrierte Schaltungen zur Realisierung von DNFs. Sie sind durch sogenannte Programmiergeräte vom Kunden (Anwender) selbst personalisierbar. PLAs besitzen eine regelmäßige Struktur und sind daher besonders für eine VLSI-Realisierung geeignet.



Literale:

$X = x_1, x_2, \ldots, x_n$ und $\overline{X} = \overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}$

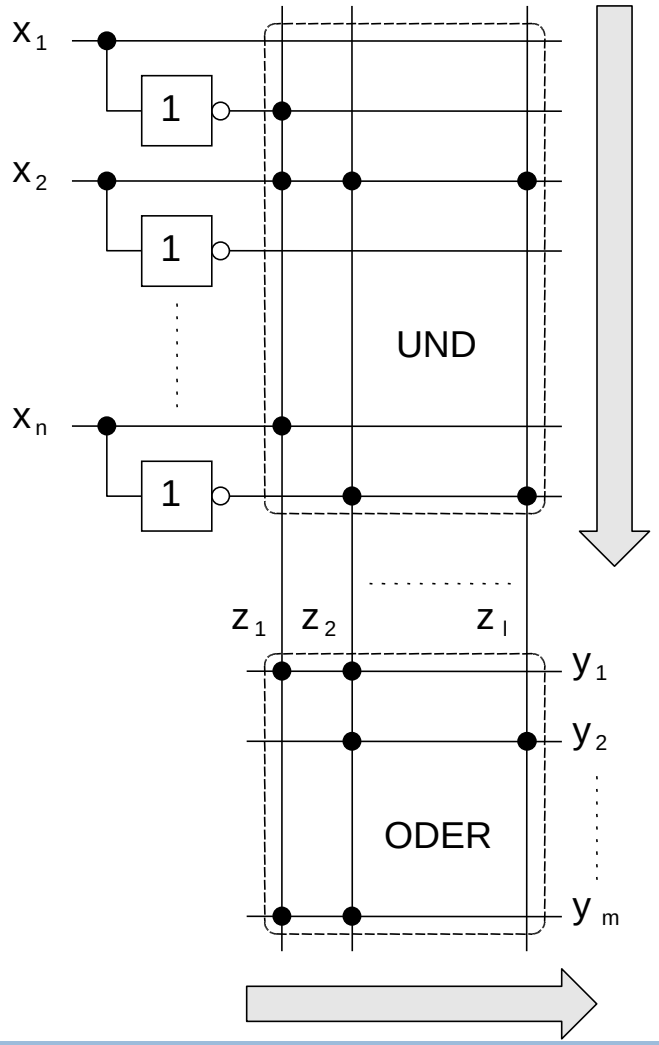
Ausgangsvektor:

Jede Komponente y_i aus $Y = y_1, y_2, \ldots, y_m$ stellt eine DNF dar.

Produkttermvektor:

Jede Komponente z_k aus $Z = z_1, z_2, \ldots, z_l$ stellt einen Konjunktionsterm der negierten oder nicht negierten Literale dar.

PLA (programmable logic arrays)



UND-Ebene (UND-Matrix):

$$z_k = \left(\bigwedge_{l \in I_{k,p}} x_l\right)\left(\bigwedge_{l \in I_{k,n}} \overline{x_l}\right)$$

mit den Indexmengen $I_{k,n}$ und $I_{k,p}$ der negierten bzw. nicht negierten Variablen

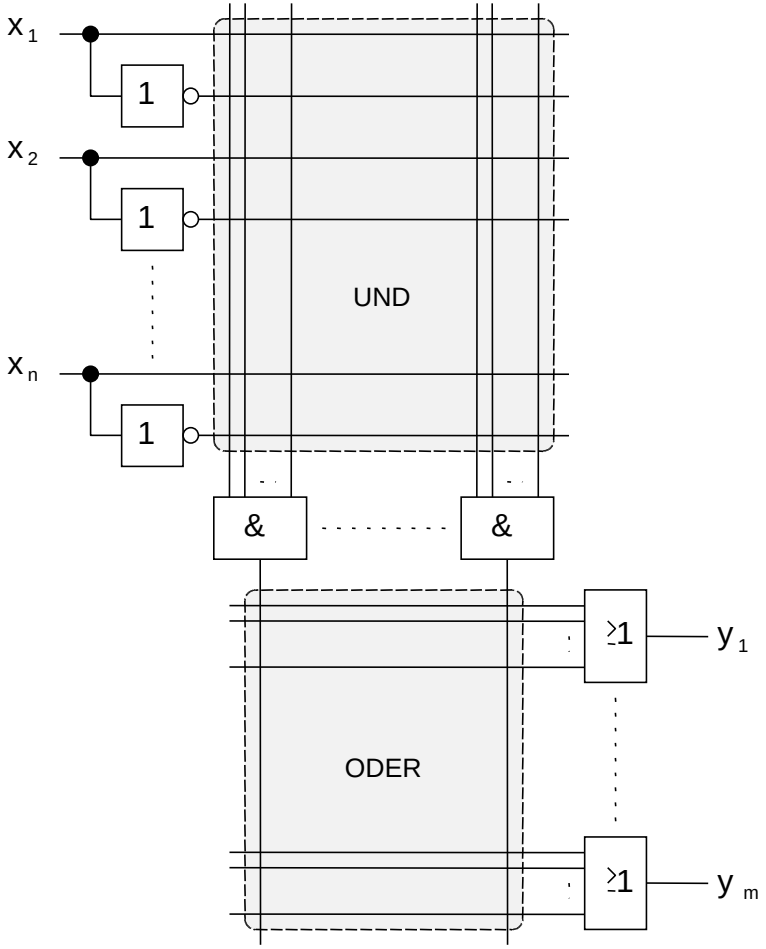
ODER-Ebene (ODER-Matrix):

$$y_i = \bigvee_{k \in I_i} z_k$$

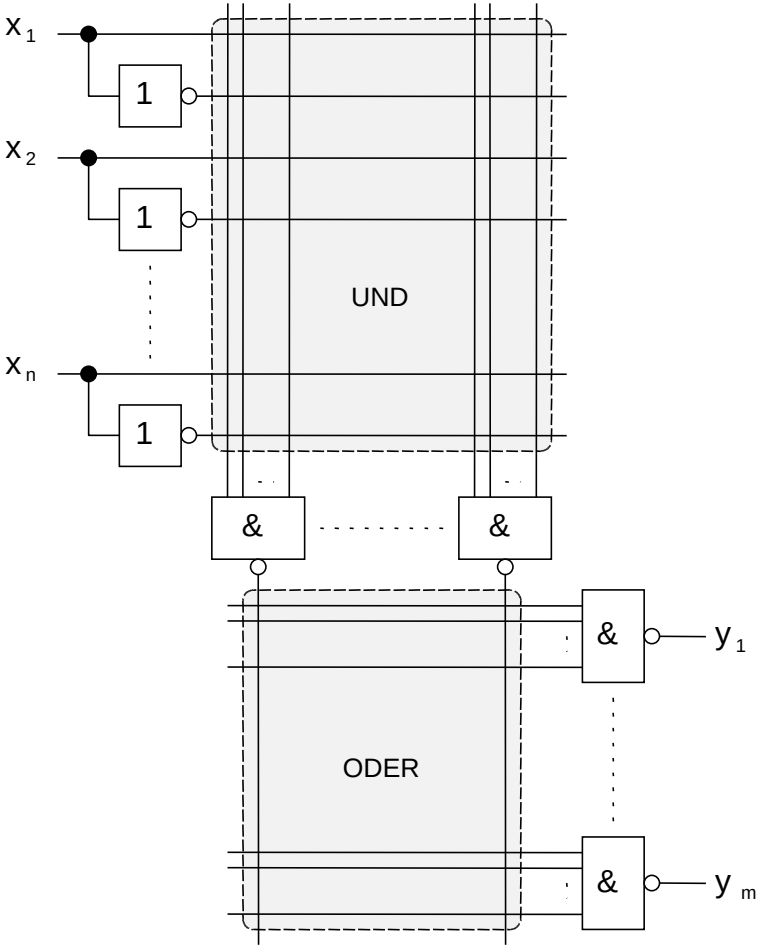
Die Personalisierung der Matrizen erfolgt durch Aktivierung der Leitungsverzweigungen, meist durch aktive Bauelemente (Transistoren).

PLA

UND/ODER-PLA:



NAND-PLA:



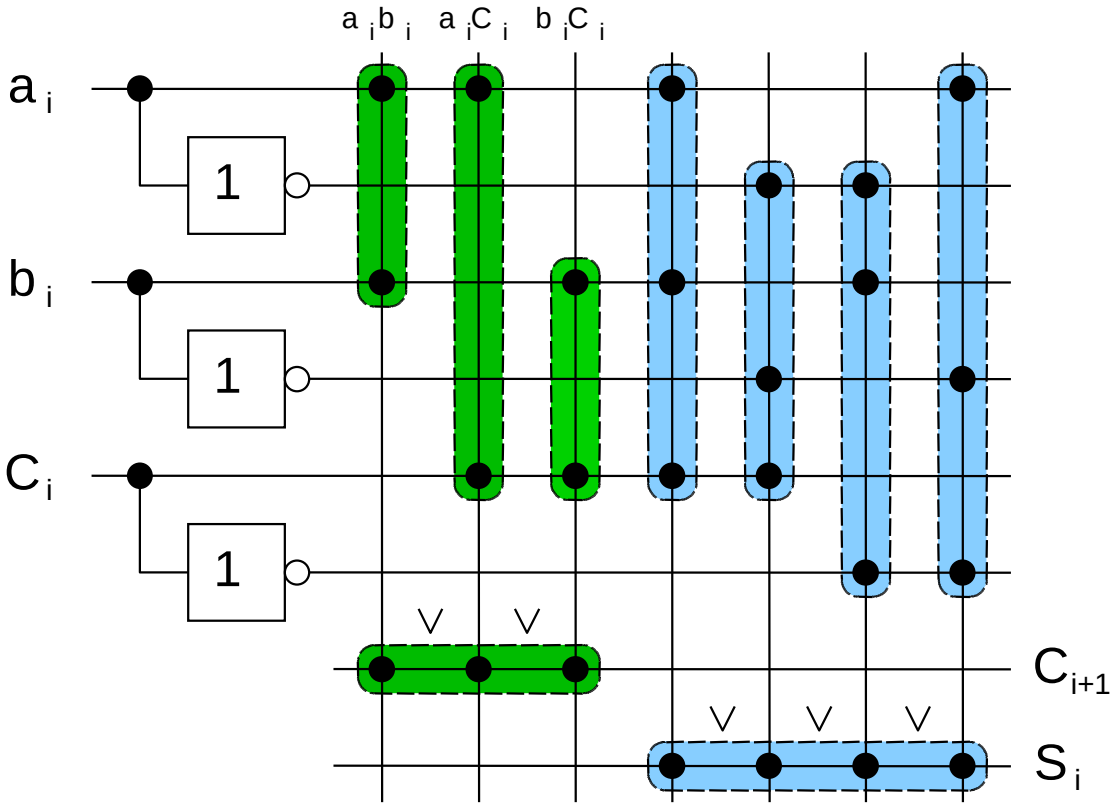
Beispiel – Volladdierer

Boolesche Funktionen:

$$C_{i+1} = a_i b_i \vee a_i C_i \vee b_i C_i$$

$$S_i = a_i b_i C_i \vee \overline{a}_i \overline{b}_i C_i \vee \overline{a}_i b_i \overline{C}_i \vee a_i \overline{b}_i \overline{C}_i$$

PLA:



PLA, PAL und ROM

Der Aufwand der Personalisierung wird geringer, wenn nur eine der beiden Matrizen programmierbar (personalisierbar) ist. Für die Realisierung Boolescher Funktionen stehen 3 personalisierbare Strukturen zur Verfügung:

- PLA

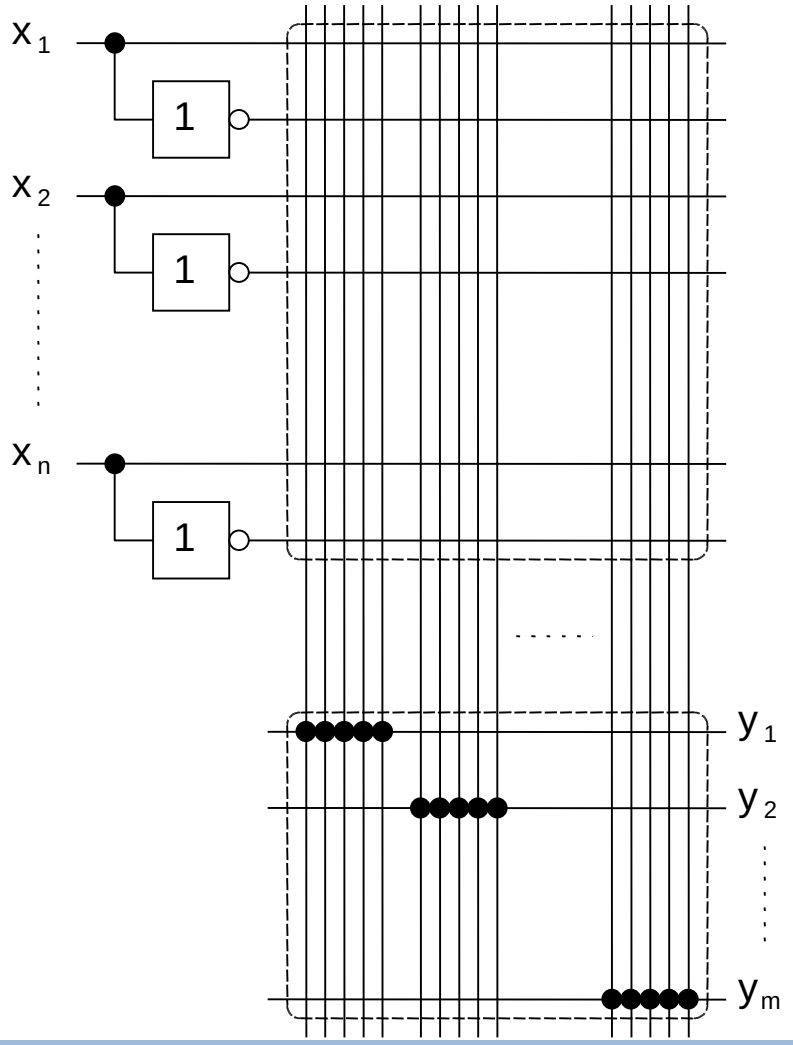
  Personalisierung: UND-/ODER-Matrix
- PAL

  Personalisierung: UND-Matrix
- Festwertspeicher (ROM)

  Personalisierung: ODER-Matrix

PAL (Programable Logic Array)

Bei einem PAL ist die UND-Matrix personalisierbar und die ODER-Matrix festgelegt.

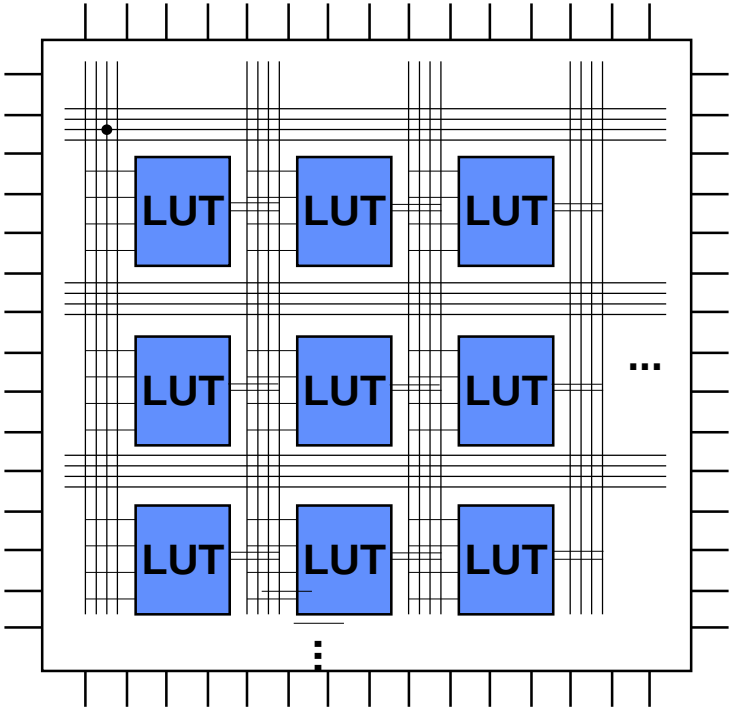


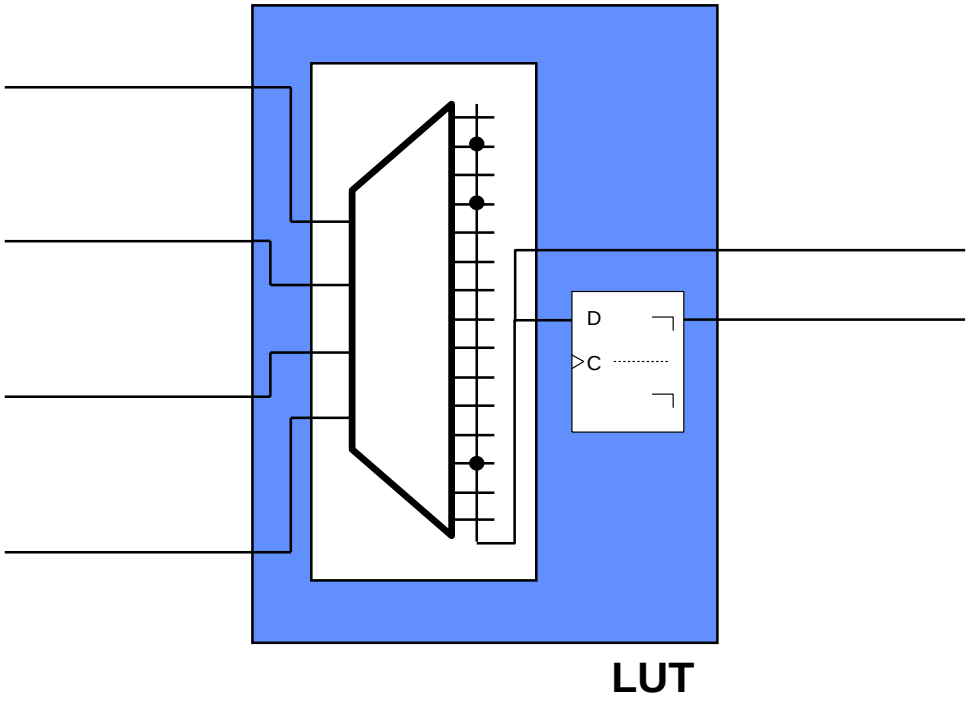
FPGA Field Programmable Gate Array

Look-Up-Table (LUT) basierter Baustein. Programmierbar über

- Oder-Matrix in der Look-UP-Table
- Verknüpfungen der Verbindungsleitungen zwischen Look-Up-Tables

Bis zu ca. 100.000 Gatteräquivalente pro Baustein

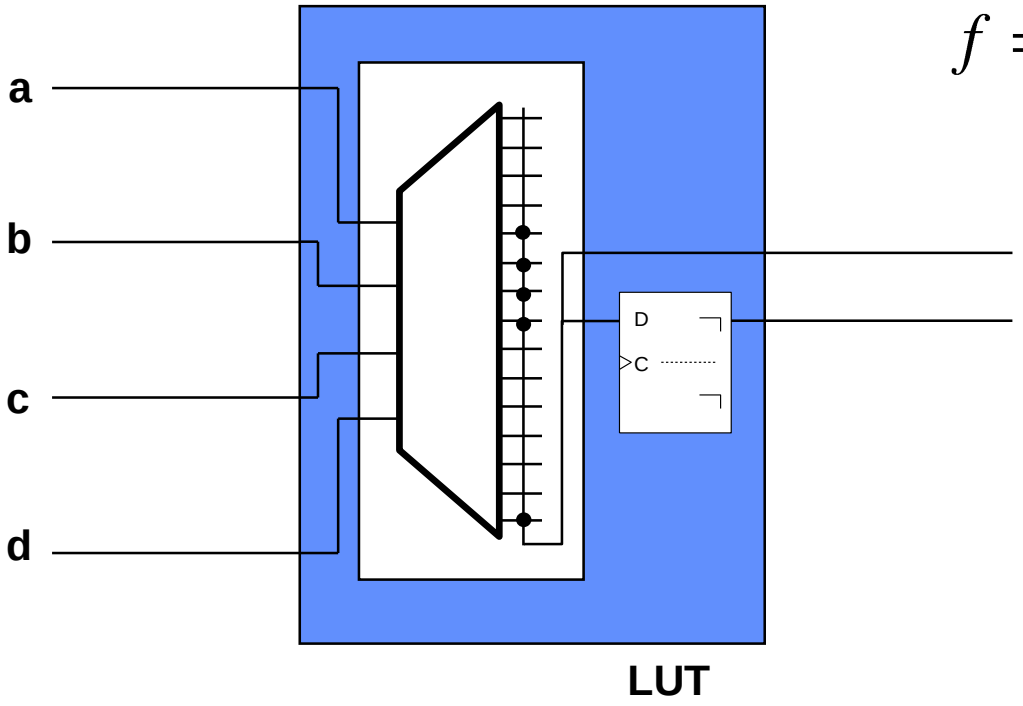




FPGA: Programmierung einer LUT

Look-Up-Table (LUT) basierter Baustein. Programmierbar über

- Oder-Matrix in der Look-UP-Table
- Das entspricht einer Wertetabelle mit 16 Eintraegen

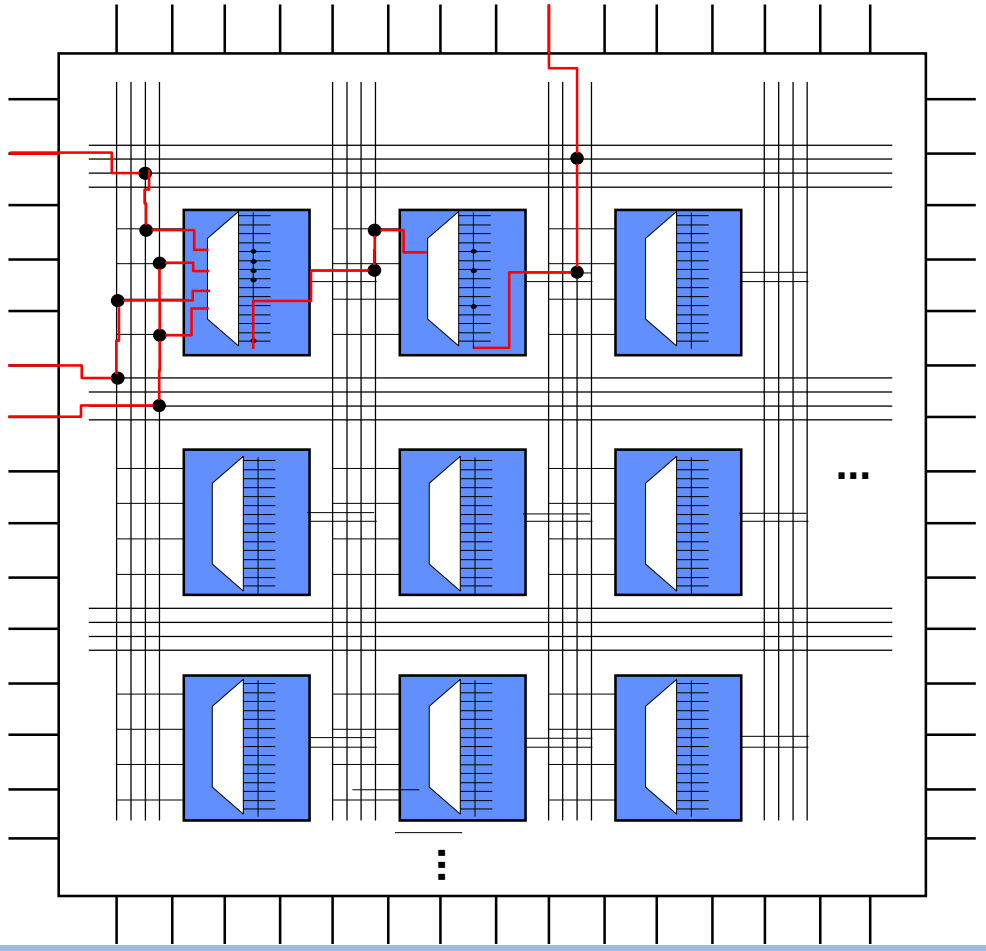


$$f = abcd \vee \overline{a}b$$

a	b	c	d	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

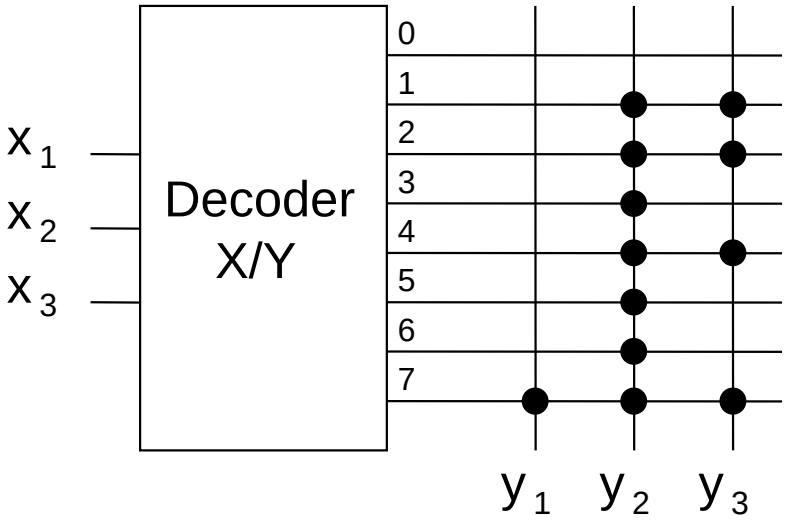
Ein programmiertes FPGA

- Verknüpfungen der Verbindungsleitungen werden programmiert
- LUT programmiert



ROM

Bei einem Festwertspeicher wird die UND-Matrix fest als Adressdecoder personalisiert.



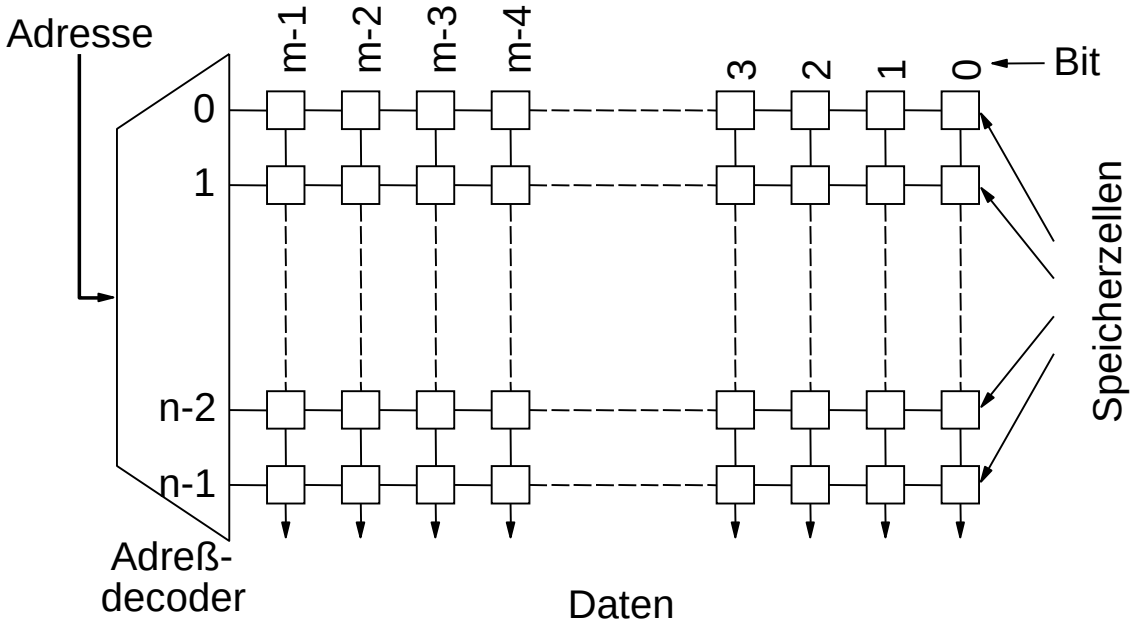
Adresse (dezimal)	x_1	x_2	x_3	y_1 $\wedge$	y_2 $\vee$	y_3 $\oplus$
0	0	0	0	0	0	0
1	0	0	1	0	1	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	0	1	1
5	1	0	1	0	1	0
6	1	1	0	0	1	0
7	1	1	1	1	1	1

RAM

Ein Schreib-/Lesespeicher (RAM) hat eine ähnliche Struktur, bei der die personalisierten Leitungsverzweigungen der ODER-Matrix durch Speicherzellen (Flipflops) ersetzt werden. Dadurch kann die Information in der ODER-Matrix jederzeit und schnell geändert werden.

Die UND-Matrix ist wie beim ROM fest als Adressdecoder personalisiert.

Aufbau eines $n \times m$-Bit Arbeitspeichers:



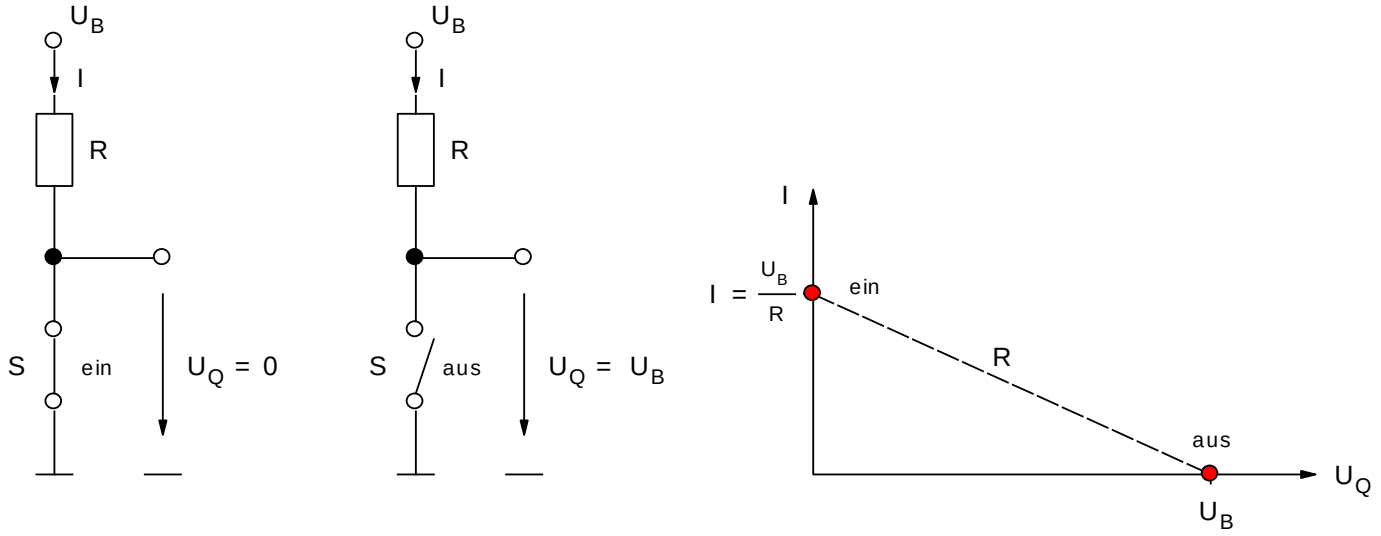
4.3 Elektrotechnische Grundlagen

Allgemein:

- In einem digitalen Datenverarbeitungssystem werden auf der physikalischen Ebene binäre Schaltvariablen mit elektronischen Schaltern nach den Gesetzen der Schaltalgebra verknüpft.
- Elektronische Verknüpfungsglieder werden aus Halbleiterbauelementen aufgebaut.
- Verknüpfungsglieder werden zu Schaltnetzen und Schaltwerken zusammengefügt.
- Schaltkreisfamilien (integrierte Schaltungen) bestehen aus standardisierten Verknüpfungsgliedern, Speichergliedern, Schaltnetzen und Schaltwerken, die aus gleichen Bauelementen und nach dem gleichen elektronischen Konzept hergestellt sind.

Modell des idealen Schalters

In der Schaltalgebra werden die binären Variablen mit Verknüpfungsgliedern aus **idealen Schaltern** verknüpft.



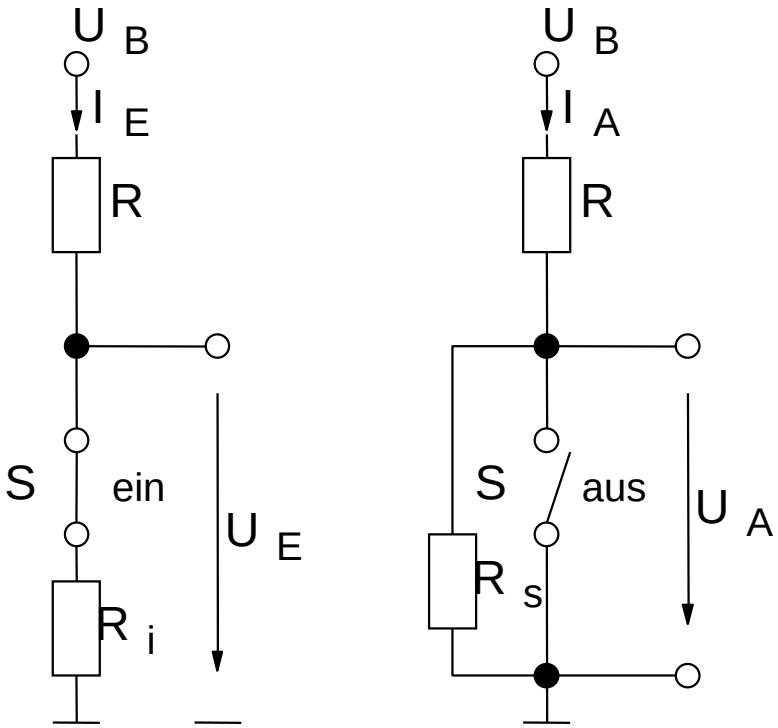
- Im Schalterzustand 'ein' ist der Innenwiderstandswert des Schalters S R_i=0. Daraus folgt $I = \frac{U_B}{R}$ und $U_Q = 0V$.
- Im Schalterzustand 'aus' ist der Sperrwiderstand des Schalters S R_i=∞. Daraus folgt $I = 0A$ und $U_Q = U_B$.

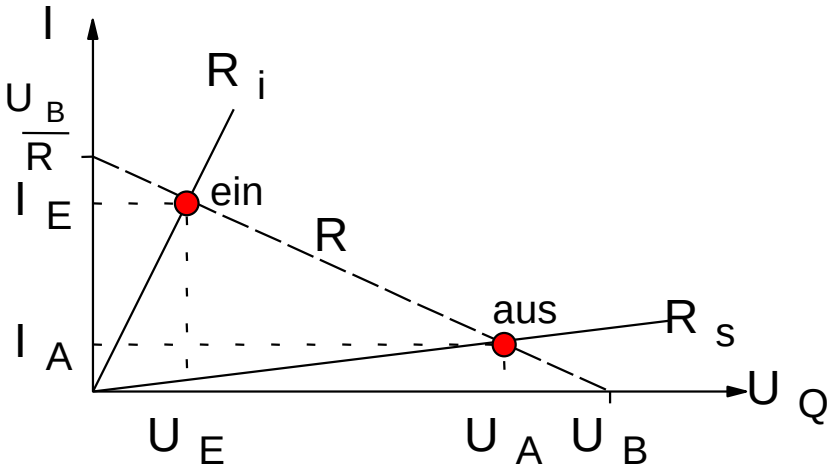
Modell des idealen Schalters

- Die Schaltwirkung folgt unmittelbar der Schaltursache, d.h. es gibt keine Zeitverzögerung.
- Die vom Schalter aufgenommene Leistung $P = U \cdot I$ ist immer Null, da entweder der Strom I (’aus’) oder die Spannung U (’ein’) gleich Null ist.

Kein realer Schalter kann diese Anforderungen erfüllen. Mit elektronischen Schaltern kommt man dem Ziel heute am nächsten. Je nach Bauelementetyp (bipolar oder unipolar) werden mehr die einen oder die anderen Eigenschaften optimal erreicht. Deshalb haben sich verschiedene Schaltkreisfamilien entwickelt.

Modell des realen Schalters





In der Schalterstellung ’ein’ liegen R und R_i in Reihe und ihre Widerstandsgeraden schneiden sich im Arbeitspunkt ein. Für Strom und Spannung gilt:

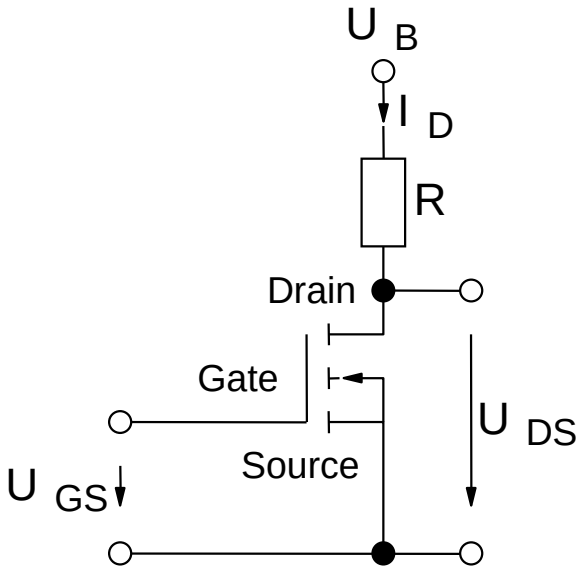
$$I_E = \frac{U_B}{R + R_i} \qquad \text{bzw.} \qquad U_E = \frac{U_B * R_i}{R + R_i}$$

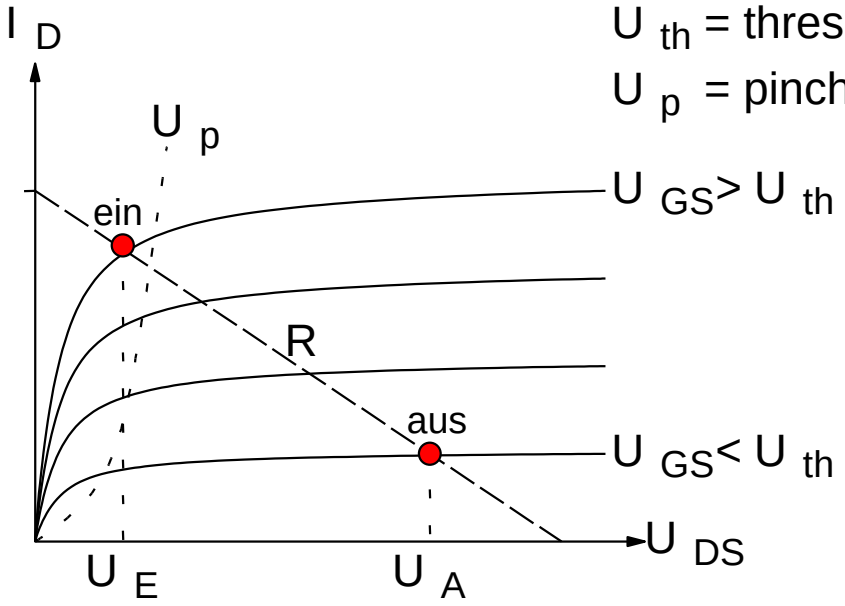
Am Schalter fällt also eine Spannung U_E ab. In der Schalterstellung ’aus’ liegen R und R_s in Reihe und ihre Widerstandsgeraden schneiden sich im Arbeitspunkt *aus*. Für Strom und Spannung gilt:

$$I_A = \frac{U_B}{R + R_s} \qquad \text{bzw.} \qquad U_A = \frac{U_B * R_s}{R + R_s}$$

Trotz Schalterstellung ’aus’ fließt ein Strom I_A. In beiden Betriebszuständen wird vom Schalter Leistung aufgenommen, weil der Strom I_A bzw. die Spannung U_E verschieden von Null sind.

MOS-FET als Schalter





U_{th} = threshold voltage

U_p = pinch off voltage

Die Schalterzustände ’ein’/’aus’ werden durch die Zustände *Transistor aus* ($U_{GS} < U_{th}$) und *Transistor ein* ($U_{GS} > U_{th}$) mit dem Schnitt der Widerstandsgeraden realisiert.

MOS-FET als Schalter

Für $U_{GS} < U_{th}$ ist die Drain-Source-Strecke gesperrt. Mit dieser Spannung wird der Transistorschalter ausgeschaltet. Der Schnittpunkt der Kennlinie für $U_{GS} < U_{th}$ mit der Widerstandsgeraden für R ist der Arbeitspunkt des Schalterzustandes 'aus'. Mit einer Spannung $U_{GS} > U_{th}$ wird die Drain-Source-Strecke leitend, der Transistor eingeschaltet.

Die Gate-Source-Spannung wird wie beim bipolaren Transistor so gewählt, daß die zugehörige Kennlinie von der Widerstandsgeraden für R im linearen Bereich geschnitten wird. Dieser Schnittpunkt ist der Arbeitspunkt des Schalterzustandes 'ein'. Wechselt die Gate-Source-Spannung zwischen $U_{GS} < U_{th}$ und $U_{GS} > U_{th}$, dann schaltet der Transistor zwischen gesperrt und leitend bzw. U_{DS} zwischen U_A und U_E.

Der Vorteil von MOS-FETs als Schalter gegenüber bipolaren Transistoren besteht darin, dass sie leistungslos am Gate angesteuert werden können.

Kenngrößen

Signalpegel:

Es werden Pegelbereiche eingeführt, die die Werte der binären Schaltvariablen darstellen. Dadurch werden die Einflüsse der Störspannung berücksichtigt. Für die Zuordnung der Pegelbereiche zu den Werten der binären Schaltvariablen gibt es zwei Möglichkeiten:

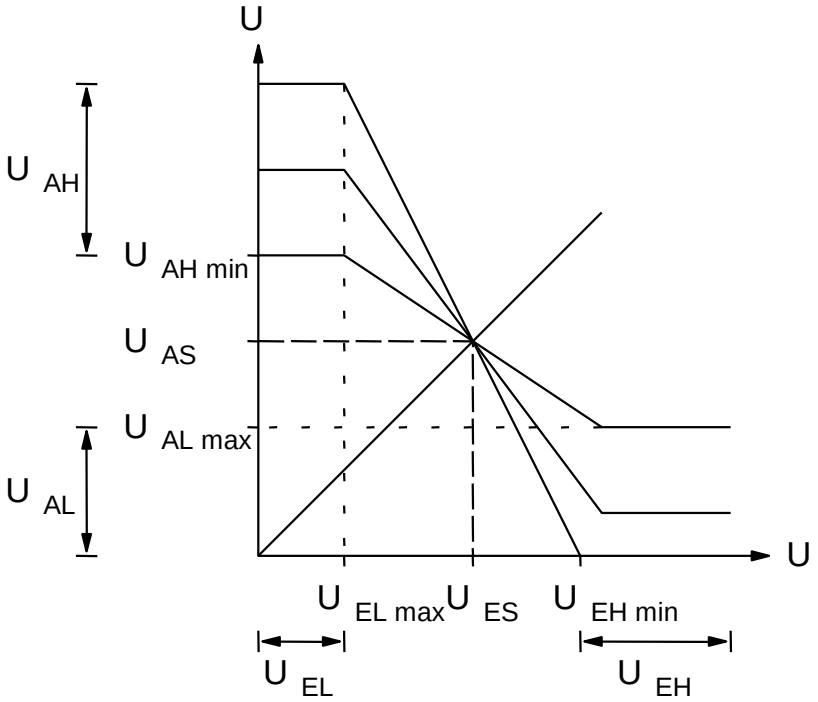
- Positive Zuordnung: $H \mathrel{\hat{=}} 1, L \mathrel{\hat{=}} 0$ (üblich)
- Negative Zuordnung: $H \mathrel{\hat{=}} 0, L \mathrel{\hat{=}} 1$

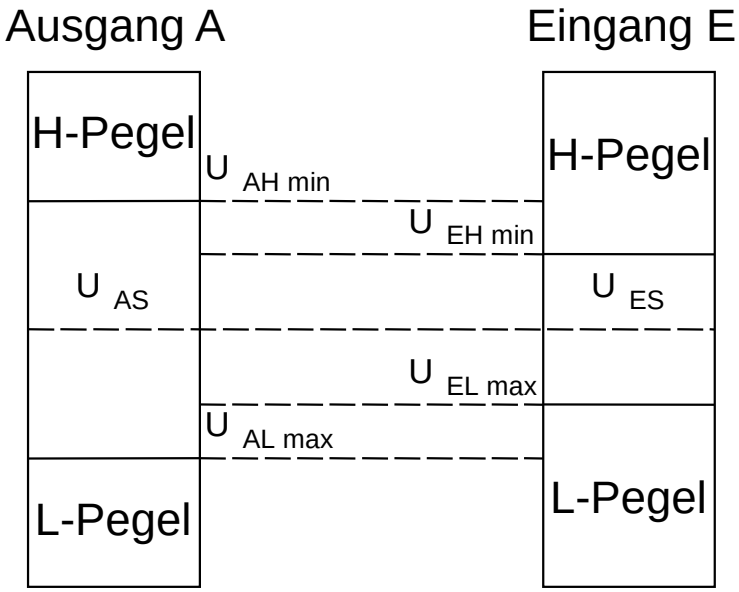
Der typische statische Störabstand U_{SS} ergibt sich aus der Differenz der Ausgangsspannung des steuernden Schaltgliedes zur Eingangsschwellspannung U_{ES} des angesteuerten Schaltgliedes.

- bei H-Pegel: $U_{SSH} = U_{AH} - U_{ES}$
- bei L-Pegel: $U_{SSL} = U_{ES} - U_{AL}$

Kenngrößen

Pegelbereiche:





Meist werden 'worst-case' Störspannungsabstände definiert:

- bei H-Pegel: $U_{SSH} = U_{AH_{min}} - U_{EH_{min}}$
- bei L-Pegel: $U_{SSL} = U_{EL_{max}} - U_{AL_{max}}$

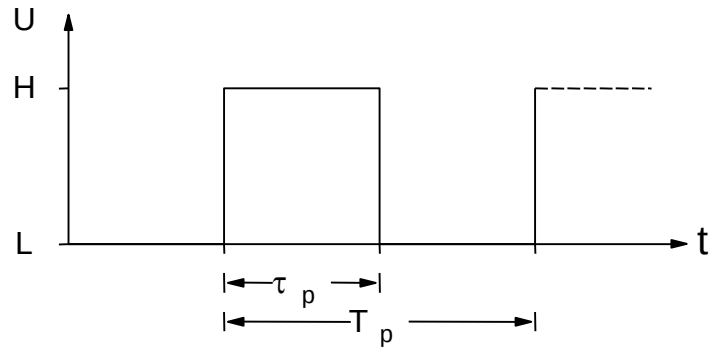
Kenngrößen

Signalübertragungszeit:

Elektronische Schalter benötigen Zeit, um von einem Schaltzustand in den anderen zu gelangen. Hauptursache für diese Zeitverzögerung ist die *kapazitive* Eigenschaft der Bauelemente; beim bipolaren Transistor hauptsächlich der Basis-Emitter *pn*-Übergang, beim unipolaren Transistor die Gate-Oxid-Substrat Schichtfolge (MOS-Kondensator).

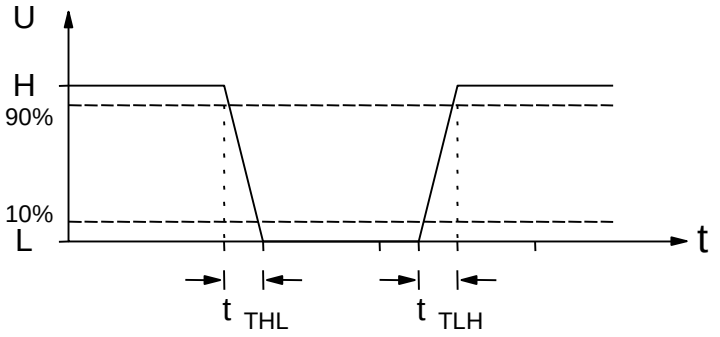
Die eigentlichen *Signalübergangszeiten* (Transition time) der Impulsflanken liegen zwischen 90% und 10% der Amplitude.

idealer Rechteckimpuls am Eingang



τ_p Pulsdauer
T_p Pulsperiode

linearisierter Ausgangsimpuls



t_{THL} Transition Time H->L
t_{TLH} Transition Time L->H

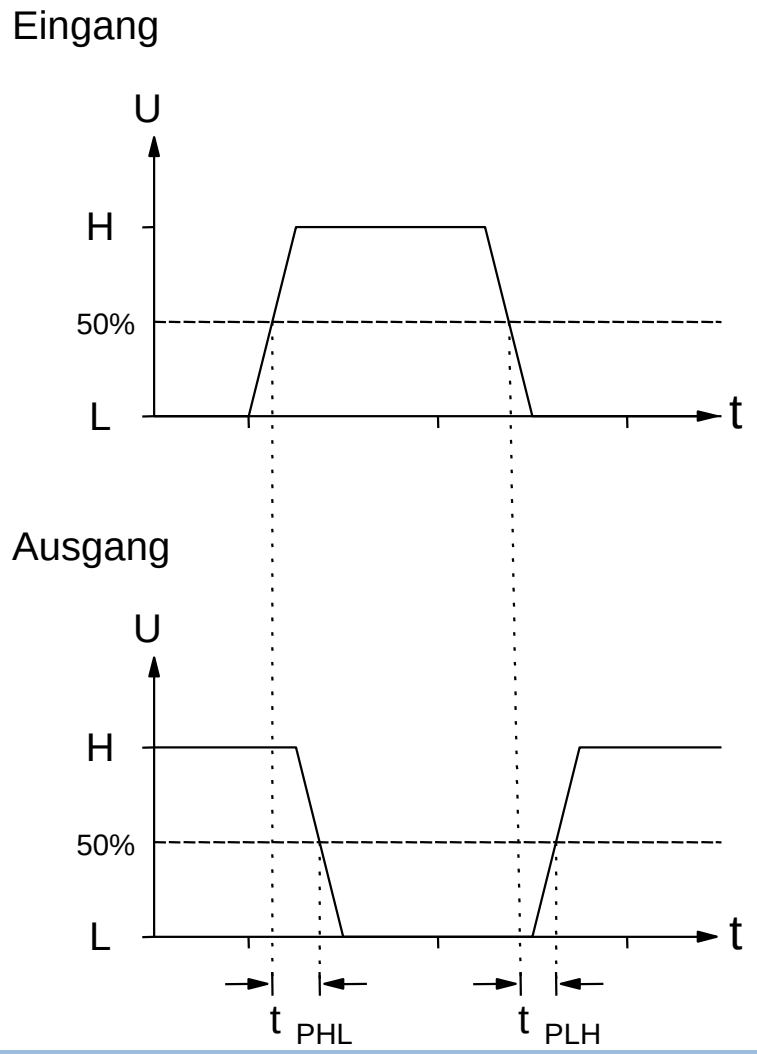
Kenngrößen

Signallaufzeit:

Die *Signallaufzeit* (Propagation delay time) gibt die Impulsverzögerung zwischen Eingangs- und Ausgangspegel an (t_{PHL} bzw. t_{PLH}). Die Messung der Signallaufzeiten wird auf die 50% Marke der Amplitude bezogen, die zwischen dem H- und dem L-Pegel liegt.

Als mittlere Signallaufzeit eines Schaltgliedes wird definiert:

$$t_P = \frac{t_{PHL} + t_{PLH}}{2}$$

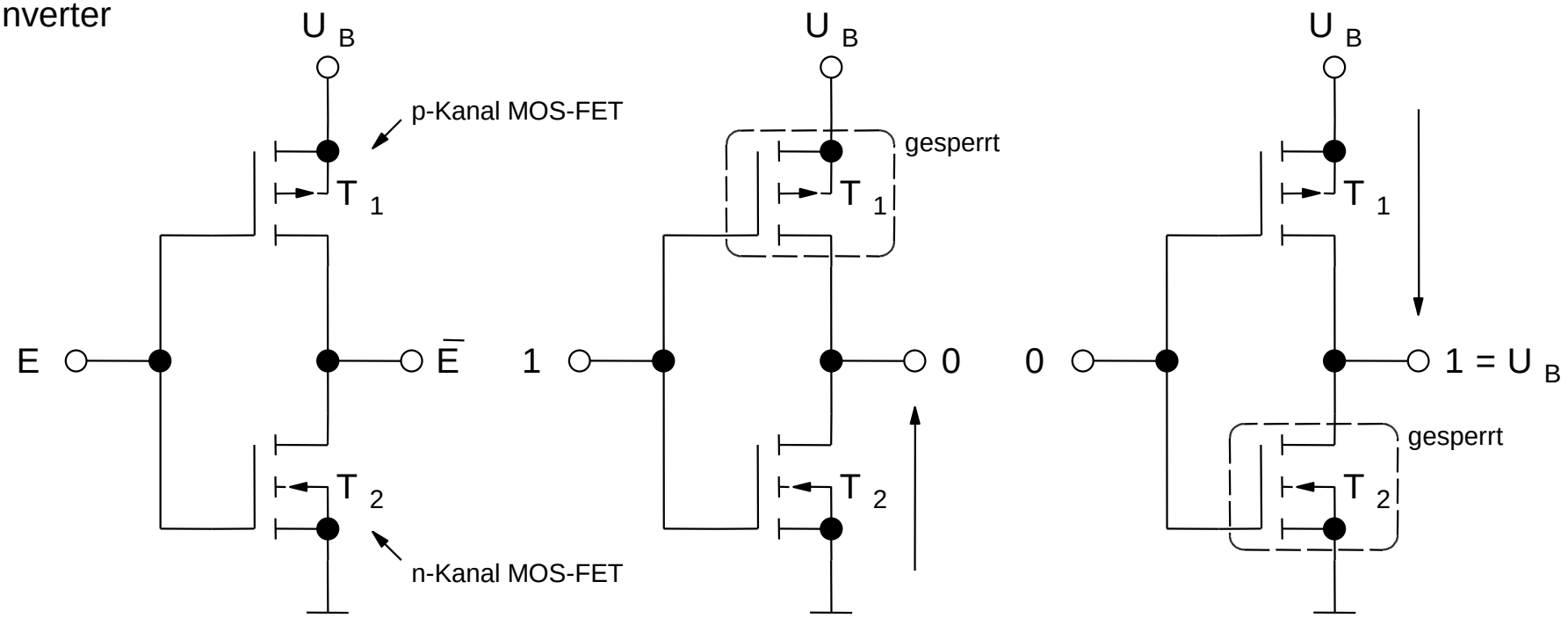


Verknüpfungsglieder: unipolare MOS-FETs

Complementary MOS (CMOS) Technik:

Verknüpfungsglieder werden in dieser Technologie aus (selbstsperrenden) NMOS- *und* PMOS-Transistoren aufgebaut.

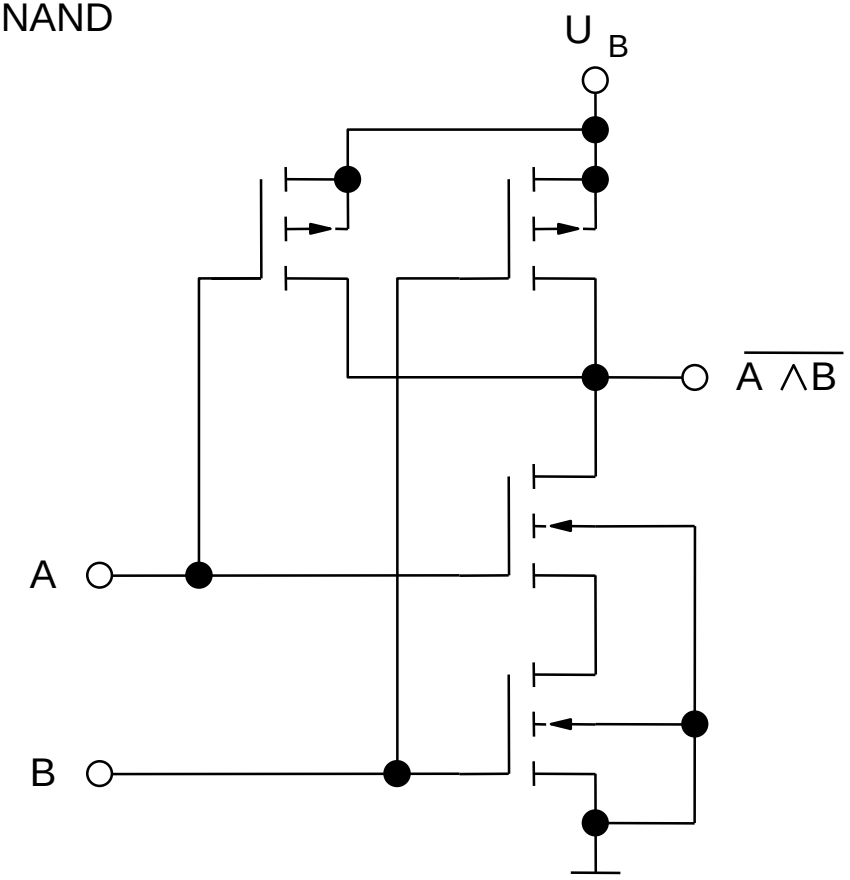
Inverter



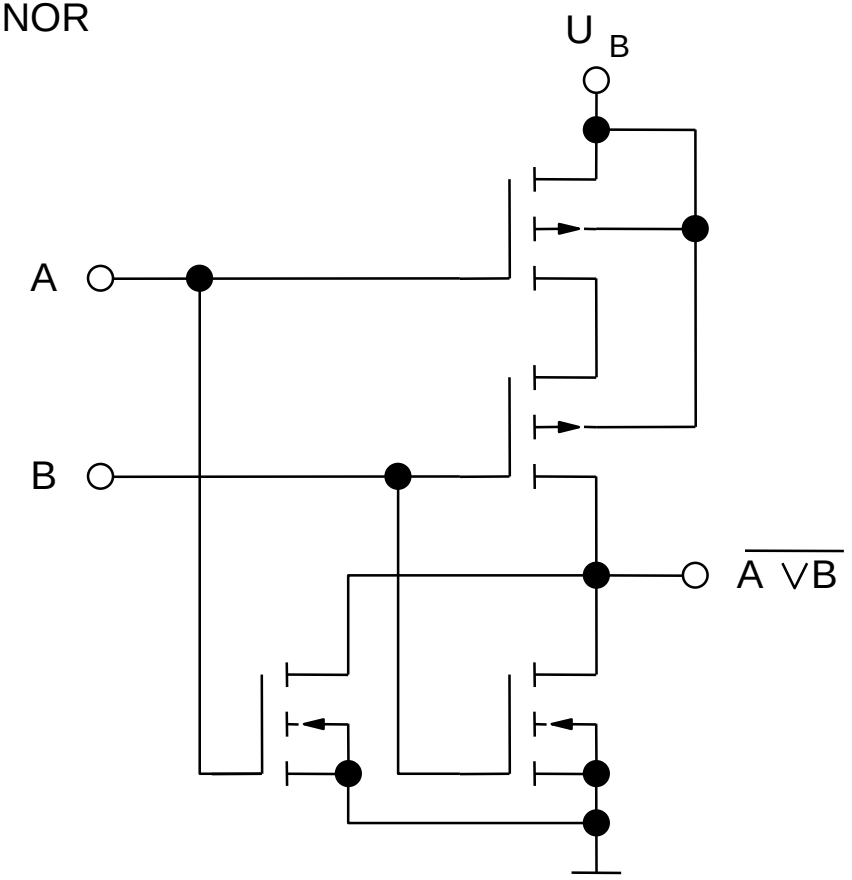
Es ist stets ein Transistor gesperrt und der andere leitend. Daher ist der Betriebsstrom und die statische Verlustleistung nahezu Null.

Verknüpfungsglieder: MOS-FETs

NAND



NOR



4.4 Zeitliches Verhalten von Schaltnetzen

Bei den bisherigen Betrachtungen wurde die Funktion eines Schaltnetzes ständig, d.h. ohne Verzögerung, ausgeführt. Jedes Signal, welches ein Gatter durchläuft, hat jedoch eine kurze, nicht vernachlässigbare Laufzeit. Diese wird durch die technologische Realisierung der Gatter hervorgerufen.

Trägheitseffekte:

Insbesondere weisen Gatter auch Trägheitseffekte auf. Diese führen dazu, dass Signaländerungen am Eingang nur dann am Ausgang wirksam werden, wenn sie eine gewisse Dauer überschreiten. Kurze Signaländerungen werden verschluckt.

Modellierung zeitlichen Verhaltens:

Viele Fehler in Schaltnetzen sind mit formalen Methoden nicht oder nur schwer zu erkennen und resultieren vor allem aus dem oben beschriebenen zeitlichen Verhalten (Verzögerung, Absorbtion) der Gatter.

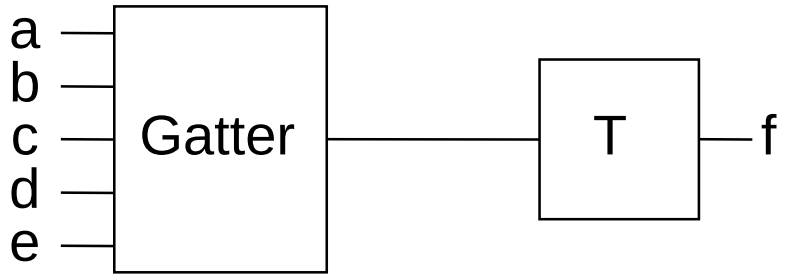
Zeitliches Verhalten von Schaltnetzen

Schaltungen (z.B. in eingebetteten Systemen) funktionieren nur dann richtig, wenn das Schaltnetz das Ergebnis innerhalb eines bestimmten Zeitintervalls berechnet hat.

Die Verzögerungszeit von Schaltnetzen lässt sich anhand von Modellen vorhersagen und optimieren. Mit diesen Modellen kann die Schaltung auch simuliert werden.

Für eine Messung wird ein Prototyp benötigt (Herstellung ist jedoch teuer und zeitaufwendig).

Modellierung des zeitlichen Verhaltens I



ideal, verzögerungsfrei

Laufzeitglied

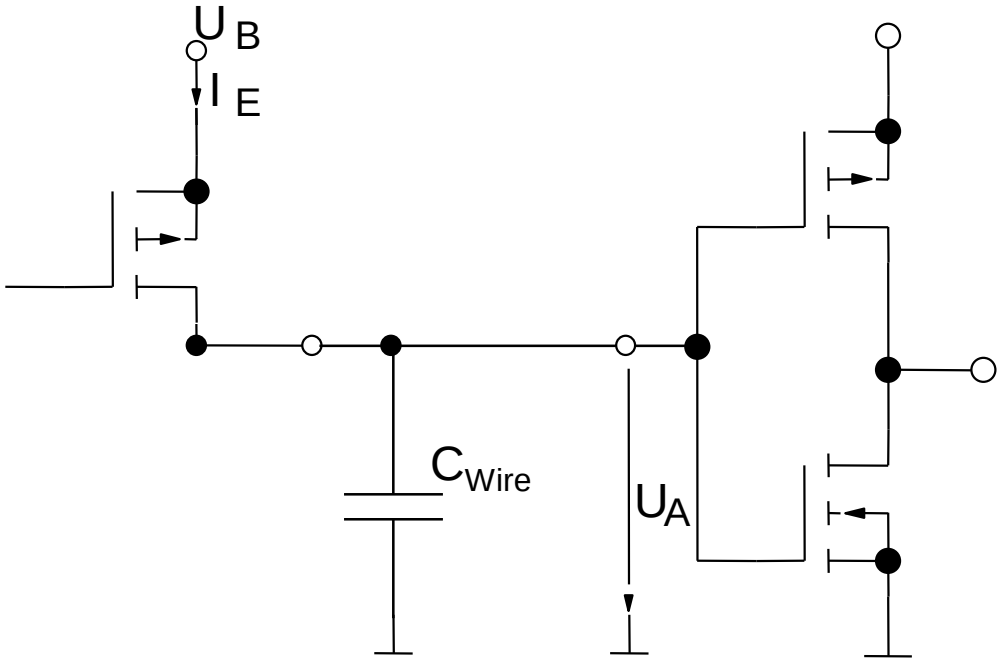
Die Verzögerungszeit ist abhängig von:

- Typ des Übergangs XX ($H \hat{=} High$, $L \hat{=} Low$)
  $HL : H \rightarrow L$, $LH : L \rightarrow H$
- Gattertyp (intrinsic delay)
- Last (*load*) am Ausgang (extrinsic delay)

  Die Last kann entweder durch andere Gatter (*Cload*) oder auch durch längere Leitungen (*Wload*) hervorgerufen sein.

Modellierung des zeitlichen Verhaltens II

Die Verzögerungszeit ist abhängig vom Umladewiderstand (Innenwiderstand des geschlossenen Transistors R_i) und der Belastung durch die Wire-Kapazität und Gate-Kapazität des nächsten Gatters. Eine gute Näherung für die Dauer des Umladevorgang ist die Zeitkonstante des RC-Gliedes.



$$T = delay \approx R_i * (C_{Wire} + C_{Load})$$

Modellierung des zeitlichen Verhaltens III

Weiter abstrahiert ist die Verzögerungszeit abhängig von:

$$T = delay = tp_{XX} + \Delta tp_{XX} * (Wload + Cload)$$

$tp \triangleq$ propagation delay

$C \triangleq$ Capacity

$W \triangleq$ Wire

Bei einer Konstruktion (Schematic) ist zunächst nur die Anzahl der Gatter am Ausgang (*Cload*) bekannt. *Wload* wird geschätzt.

Standardzellenbibliothek ECPD15

NAND4-Gatter: $f = \overline{a \wedge b \wedge c \wedge d}$

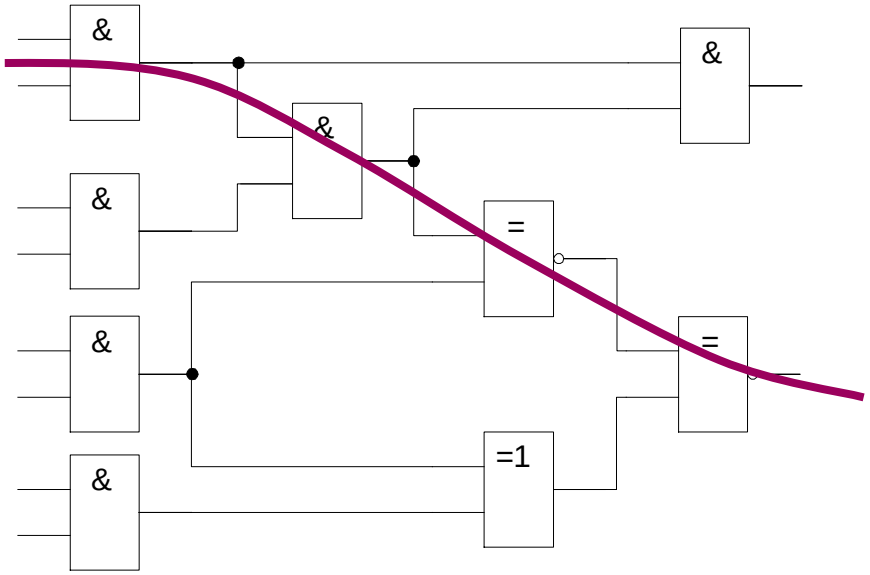
Parameter	from	to	Value	min	typ	max	mil	unit
Size			32.0*76.0					μm^2
Cin_a			0.15					pF
Cin_b			0.15					pF
Cin_c			0.14					pF
Cin_d			0.14					pF
$Fanout_f$			0.89					pF
total cap			0.66					pF
transistors			8					
tp_{lh}	any	f		0.38	0.90	1.77	2.14	ns
tp_{hl}	any	f		0.31	0.74	1.45	1.75	ns
Δtp_{lh}	any	f		0.49	1.17	2.30	2.78	ns/pF
Δtp_{hl}	any	f		0.56	1.34	2.64	3.19	ns/pF

Kritischer Pfad

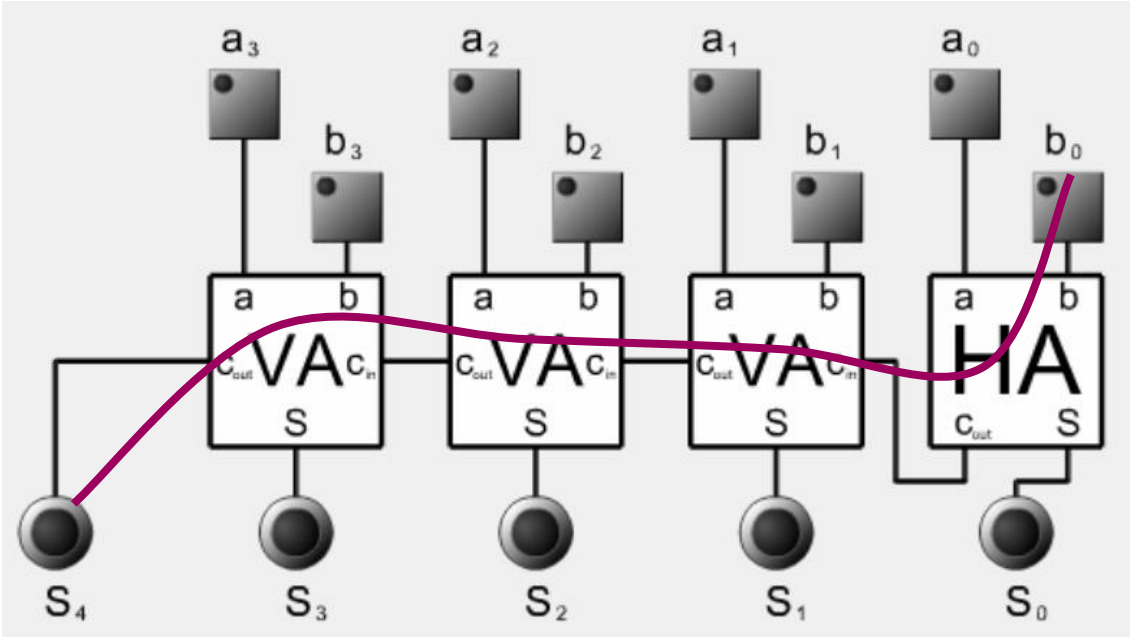
Eine obere Abschätzung für die Verzögerungszeit einer kombinatorischen Schaltung liefert die Bewertung der Laufzeit über den kritischen Pfad:

Annahme: Jedes Gatter hat eine einheitliche Verzögerung, die Verzögerung auf den Leitungen werde vernachlässigt.

Dann ist der **kritische Pfad** der längste Pfad, der von den Eingängen bis zum Ausgang der Schaltung durchlaufen werden kann.

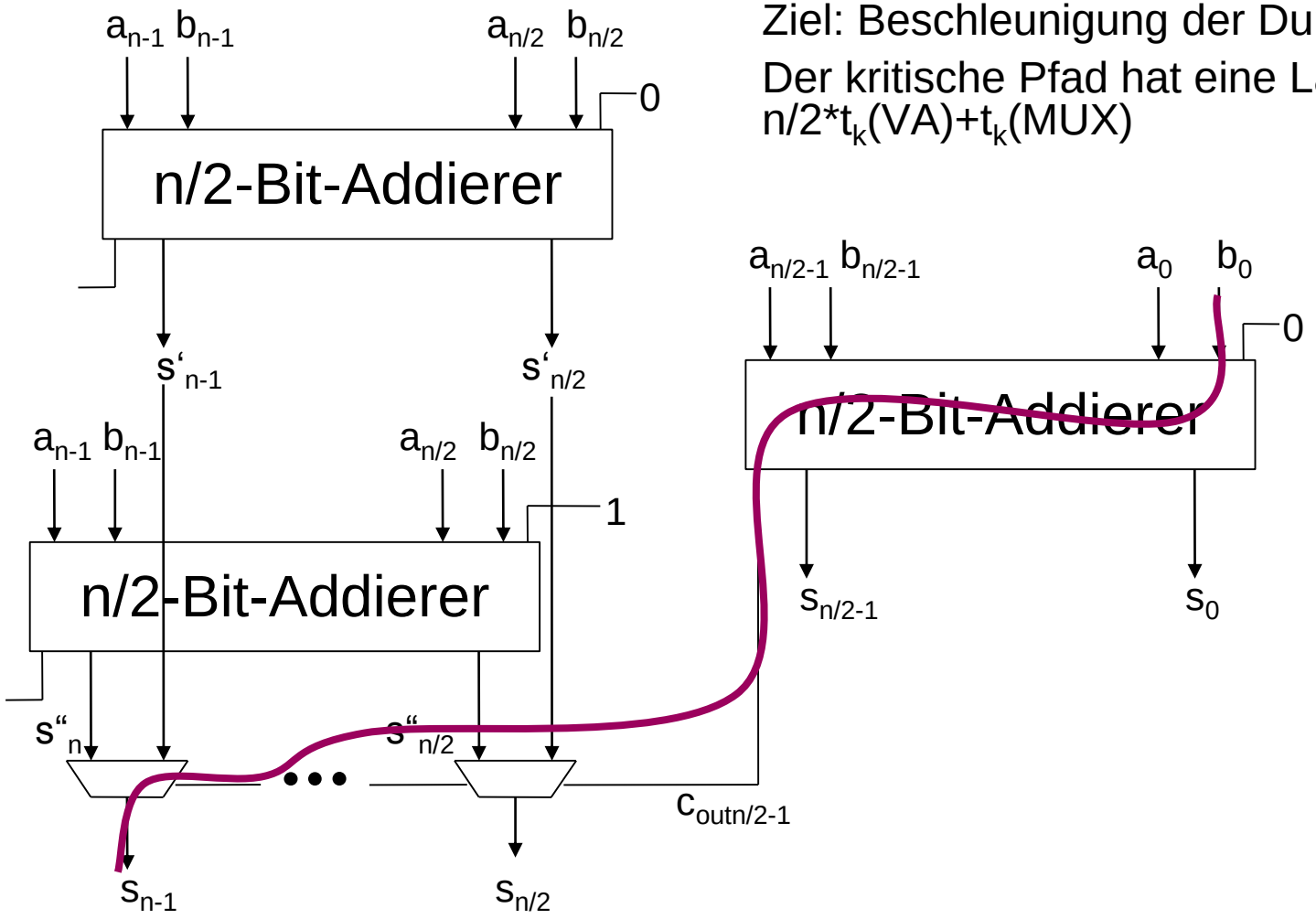


n-Bit Ripple-Carry-Addierer



Der kritische Pfad hat eine Länge von $n*t_k(VA)$

Carry-Select-Addierer



Ziel: Beschleunigung der Durchlaufzeit

Der kritische Pfad hat eine Länge von $n/2*t_k(VA)+t_k(MUX)$

Träges Laufzeitglied

Bei den bisherigen Betrachtungen wurde der Verzögerungseffekt *T* durch ein ideales Laufzeitglied

$$L_i(x(t), T) = x(t - T)$$

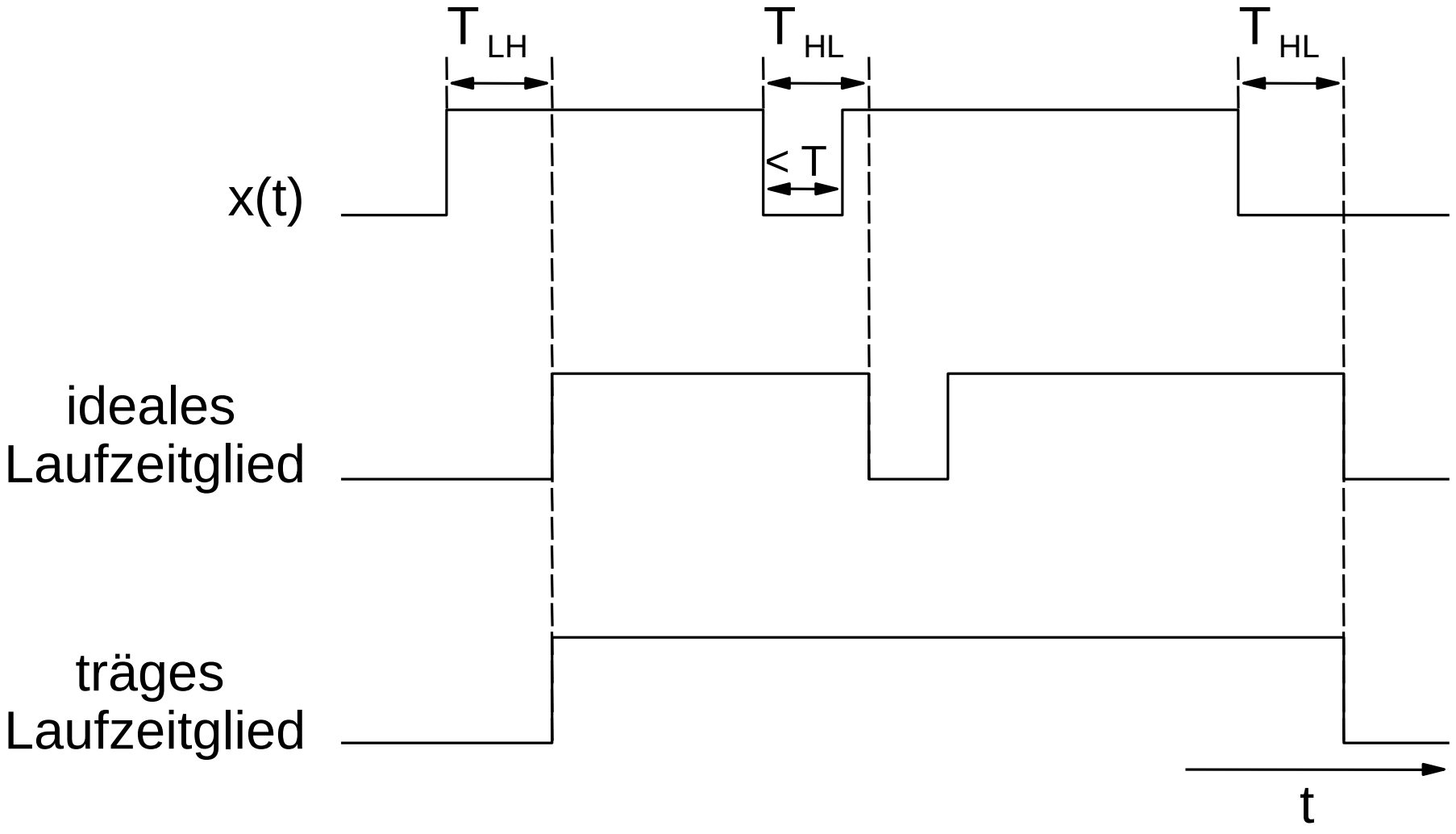
ausgedrückt. Ein träges Laufzeitglied ist ein nichtideales Laufzeitglied, bei dem die Trägheitseffekte berücksichtigt werden. Sei $x(t)$ ein ereignisdiskretes Signal der Form:

$$x(t) := x(t_i) : t_i \leq t < t_{i+1} \qquad i = 1, 2, \ldots$$

Ein träges Laufzeitglied läßt sich wie folgt beschreiben:

$$L_{tr}(x(t), T) = \begin{cases} x(t - T) & t_i - t_{i-1} \geq T \text{ oder} \\ & t - t_i \geq T \\ L_{tr}(x(t_{i-1}), T) & t_i - t_{i-1} < T \text{ und} \\ & t - t_i < T \end{cases}$$

Träges Laufzeitglied



4.5 Hazards (Gefahr) in Schaltnetzen

Hazards stellen eine weitere wichtige Fehlerquelle dar.

Beispiel:

$$f = ac \vee \overline{a}b$$

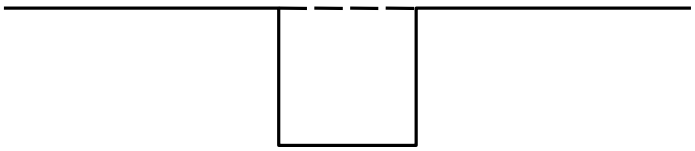
Für $b = 1$ und $c = 1$ ist unabhängig von a die Funktion $f = 1$. Bei einem Wechsel von a ($0 \rightarrow 1$ oder $1 \rightarrow 0$) und unterschiedlichen Gatterlaufzeiten T_2 und T_3 springt der Funktionswert kurzzeitig auf 0 und im Anschluss wieder auf 1. Dieses Verhalten wird Hazard (Risiko, Gefahr) genannt.

Hazards

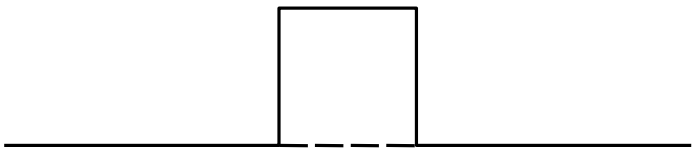
Statische Hazards:

Ein statischer Hazard liegt vor, wenn der Ausgang konstant bleiben sollte, aber kurzzeitig einen anderen Wert annimmt.

Ein statischer 0-Hazard liegt vor, wenn der Ausgang eigentlich konstant 0 sein sollte; ein statischer 1-Hazard liegt vor, wenn der Ausgang konstant 1 sein sollte.



Statischer 1-Hazard



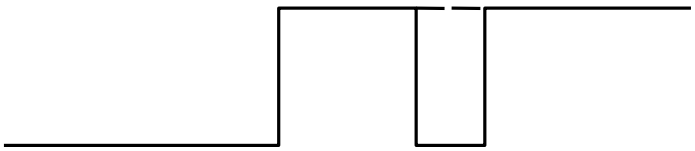
Statischer 0-Hazard

Hazards

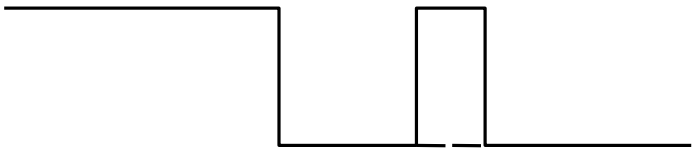
Dynamische Hazards:

Ein dynamischer Hazard liegt vor, wenn der Ausgang bei einem Übergang vor dem Einstellen auf den endgültigen Wert noch einige Male andere Werte annimmt.

Ein dynamischer 0-1-Hazard liegt vor, wenn dies beim Wechsel von 0 nach 1 geschieht; ein dynamischer 1-0-Hazard liegt vor, wenn dies beim Wechsel von 1 nach 0 geschieht.



Dynamischer 0-1-Hazard



Dynamischer 1-0-Hazard

Beispiel Dynamischer 0-1-Hazard

Hazards

Funktionshazards:

Funktionshazards sind schaltungsunabhängige Eigenschaften einer Booleschen Funktion. Sie treten auf, wenn sich mehrere Eingänge nicht gleichzeitig, sondern nacheinander ändern.

Eine Boolesche Funktion $f(x_1, \ldots, x_n)$ hat einen statischen Funktionshazard für den Übergang von $X^a = (\underline{x}^a{}_1 ; : : : ; \underline{x}^a{}_n)$ nach $X^b = (\underline{x}^b{}_1, \ldots, \underline{x}^b{}_n)$, wenn:

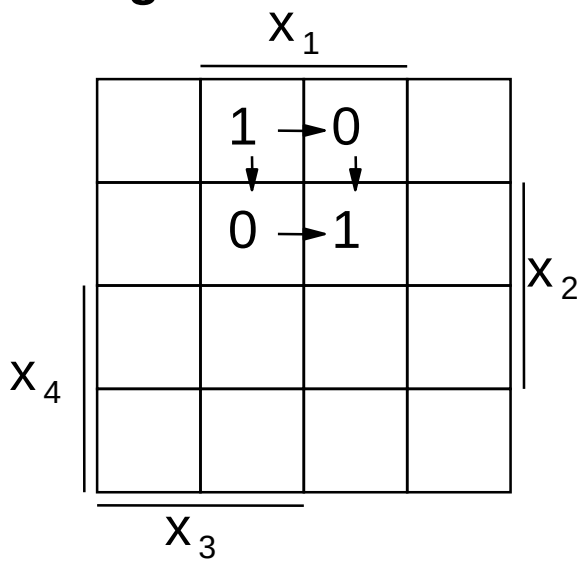
1. $f(X^a) = f(X^b)$

2. $\exists X^* = (\underline{x}^*_1, \ldots, \underline{x}^*_n)$ mit $\underline{x}^*_i = \underline{x}^a_i$ oder $\underline{x}^*_i = \underline{x}^b_i$ $(i = 1, \ldots, n)$ für das $f(X^*) \neq f(X^a)$.

Funktionshazards

Erkennen von Funktionshazards im KV-Diagramm:

$$X_1 = (x_1, \overline{x_2}, x_3, \overline{x_4})$$
$$X_2 = (x_1, x_2, \overline{x_3}, \overline{x_4})$$
$$X_* = (x_1, \overline{x_2}, \overline{x_3}, \overline{x_4}) \text{ oder}$$
$$(x_1, x_2, x_3, \overline{x_4})$$



Funktionshazards lassen sich nur durch Änderung der Booleschen Funktion vermeiden. Man kann jedoch auch dafür sorgen, dass die Änderung der Eingangsvariablen in einer bestimmten Reihenfolge geschieht.

Schaltungshazards

Schaltungshazards (auch: logische Hazards) entstehen durch die Signallaufzeiten in einzelnen Gattern einer Implementierung.

Definition:

Ein Schaltungshazard in einem Schaltnetz S, welches die Boolesche Funktion f realisiert, liegt vor, wenn:

1. f keinen Funktionshazard für den Übergang $a \rightarrow b$ besitzt
2. während des Wechsels von a nach b am Ausgang von S ein Hazard beobachtbar ist.