

## 3.3.3 Spezielle Strukturen

---

Anstelle aus logischen Gattern (UND, ODER, NICHT, NAND, NOR, ... usw.) lassen sich Schaltnetze auch mit komplexeren Standardbausteinen realisieren.

### ➔ Einfachere und oft flexiblere Realisierung

➤ Minimierung bedeutet hierbei dann nicht die Reduktion von Gattern.

➔ Es muss die Anzahl und Größe komplexer Bausteine reduziert werden.



# Multiplexer

---

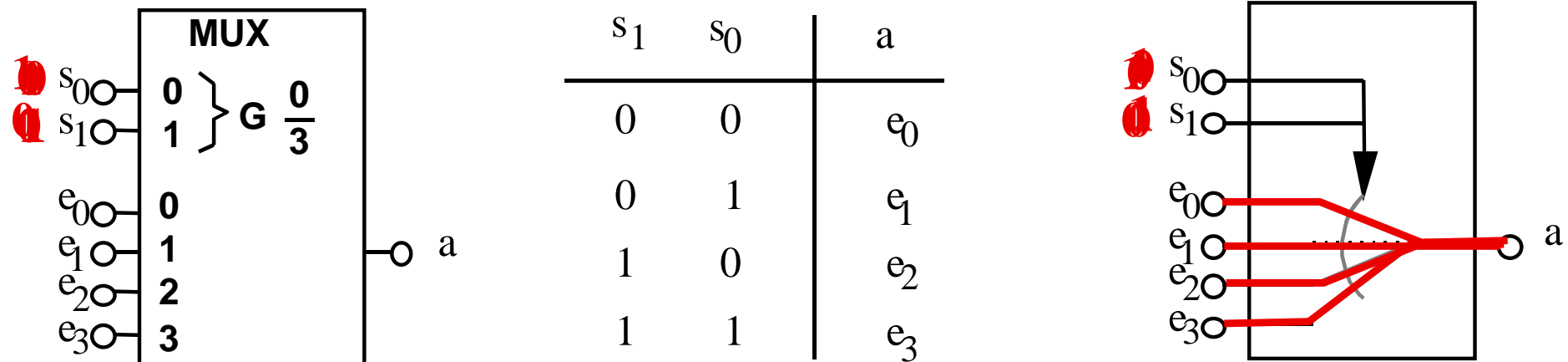
Ein **Multiplexer** (Abk.: **MUX**) ist ein Baustein mit mehreren Eingängen und einem Ausgang, wobei über  $n$  Steuerleitungen einer der  $2^n$  Eingänge auf den Ausgang geschaltet wird.

Multiplexer werden nach ihrer Größe als  **$2^n$ : 1- Multiplexer** (alternativ als **1-aus- $2^n$ - Multiplexer**) klassifiziert.



# Multiplexer

Schaltbild und logisches Verhalten eines 4:1-Multiplexers

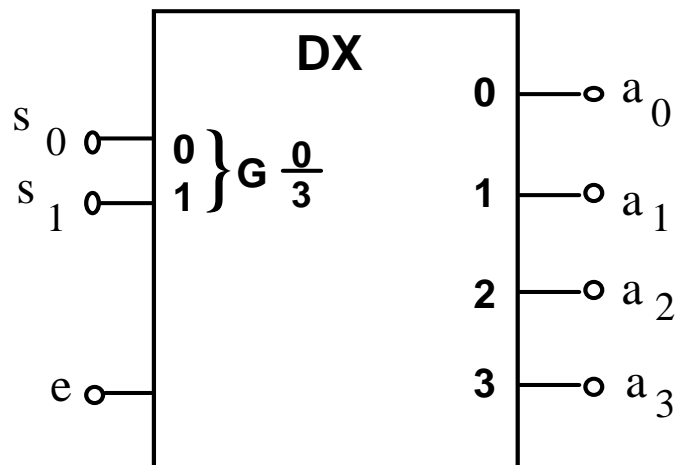


**Steuerung der Signalpfade**

# Demultiplexer / Dekoder

Der zum Multiplexer korrespondierende Baustein, der einen Eingang abhängig von  $n$  Steuerleitungen auf einen von  $2^n$  Ausgängen schaltet, heißt **Demultiplexer**.

## Beispiel:



$s_1$	$s_0$	$a_0$	$a_1$	$a_2$	$a_3$
0	0	e	0	0	0
0	1	0	e	0	0
1	0	0	0	e	0
1	1	0	0	0	e

Schaltbild und logisches Verhalten eines 1-auf-4-Demultiplexers

# Demultiplexer / Dekoder

---

- Der Demultiplexer hat einen Enable-Eingang  $e$  sowie  $n$  Eingänge  $s_i$  für eine Dualziffer, die an den  $2^n$  Ausgängen  $a_j$  dekodiert bereitgestellt wird.
  - Enable-Eingang ( $e = 0$ ), dann liegen alle Ausgänge auf  $0$
  - Sonst wird eine 2-bit-Zahl dekodiert, z. B. wird bei Anlegen der Zahl 2 ( $s_1 = 1, s_0 = 0$ ) der Ausgang  $a_2 = 1$  und alle anderen Ausgänge bleiben  $0$ .

Der Demultiplexer wird deshalb auch Dekoder genannt.



# Realisierung logischer Funktionen durch Dekoder

- Ein Dekoder erzeugt alle  $2^n$  Minterme seiner  $n$  Steuervariablen.

**Beispiel:** Berechnung der Anzahl Einsen:  $f(a,b,c) = a + b + c$

Summe

$$s = a \leftrightarrow b \leftrightarrow c$$

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

$$= \text{MINt}(1,2,4,7)$$

Übertrag

$$\ddot{u} = a b \vee a c \vee b c$$

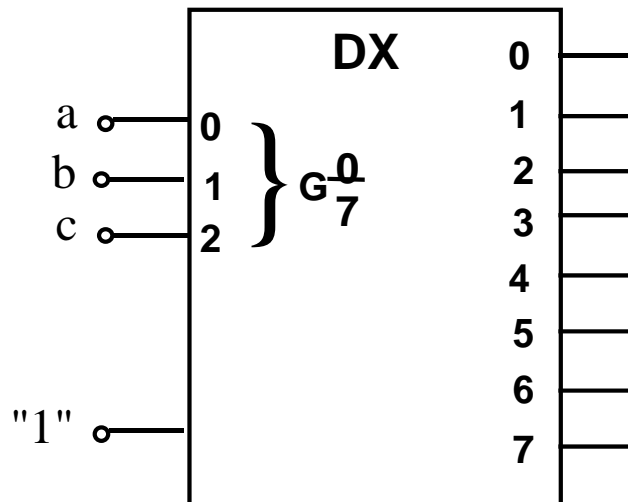
$$= \text{MINt}(3,5,6,7)$$

c	b	a	f	$\ddot{u}$	s
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	2	1	0
1	0	0	1	0	1
1	0	1	2	1	0
1	1	0	2	1	0
1	1	1	3	1	1

# Realisierung

$$\begin{aligned} s &= a \leftrightarrow b \leftrightarrow c \\ &= a \bar{b} \bar{c} \vee a b \bar{c} \vee a b c \vee \bar{a} \bar{b} c \\ &= \text{MINt}(1,2,4,7) \end{aligned}$$

$$\begin{aligned} \ddot{u} &= a b \vee a c \vee b c \\ &= \text{MINt}(3,5,6,7) \end{aligned}$$



# Realisierung mittels Speicherbausteinen

---

Bei den bisher behandelten Bausteinen (Gatter, Multiplexer, Dekoder) war die Funktion fest vorgegeben.

## ➡ **festverdrahtete Logik**

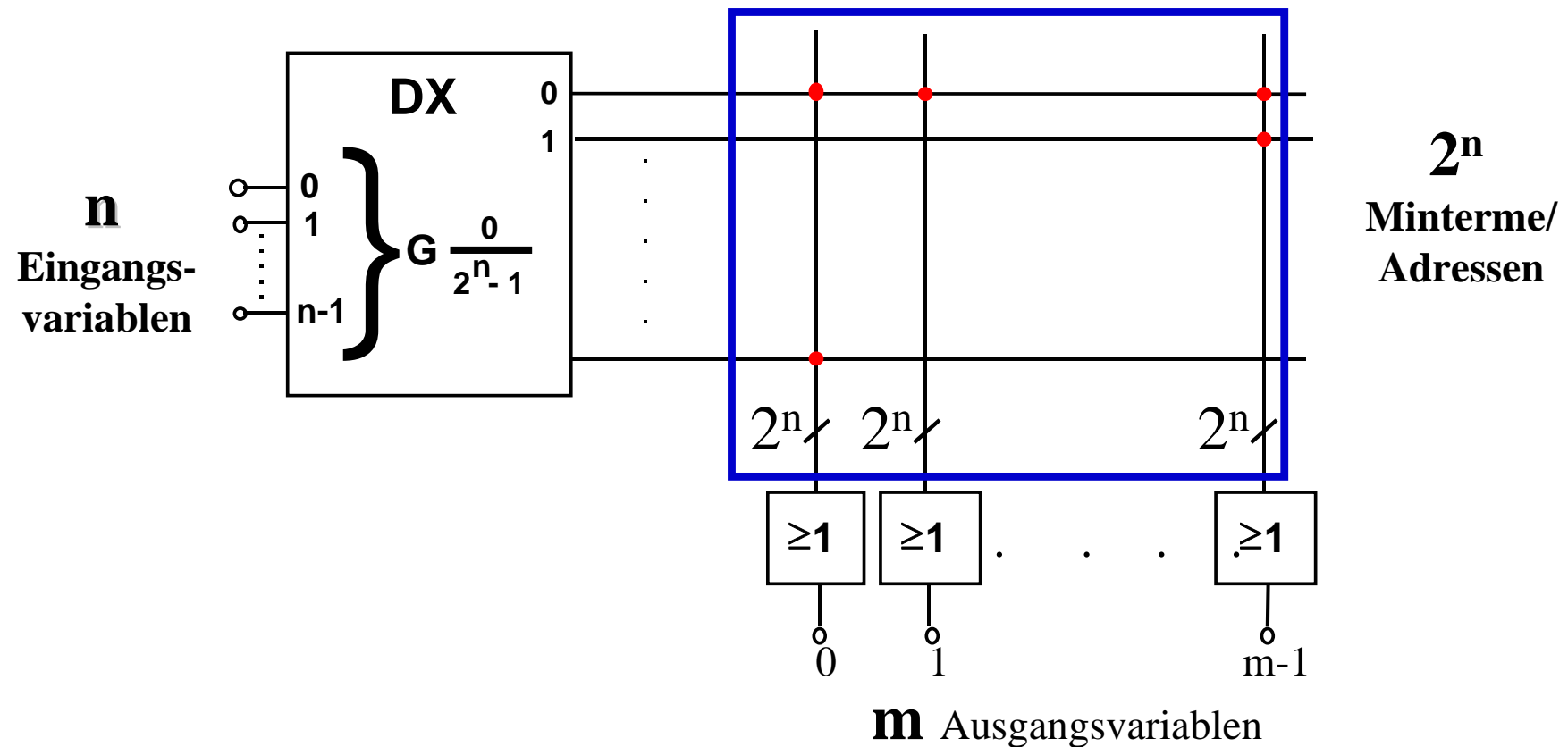
Höherintegrierte Verknüpfungsbausteine müssen die Flexibilität bieten, an viele verschiedene Anwendungen anpassbar zu sein. Diese Anpassung wird als **Personalisierung** oder als **Programmierung** bezeichnet.

## ➡ **mikroprogrammierte Logik**



# Schematischer Aufbau eines Speicherbausteins

Speicheranordnung, in der beliebige Funktionstabellen abgelegt werden.



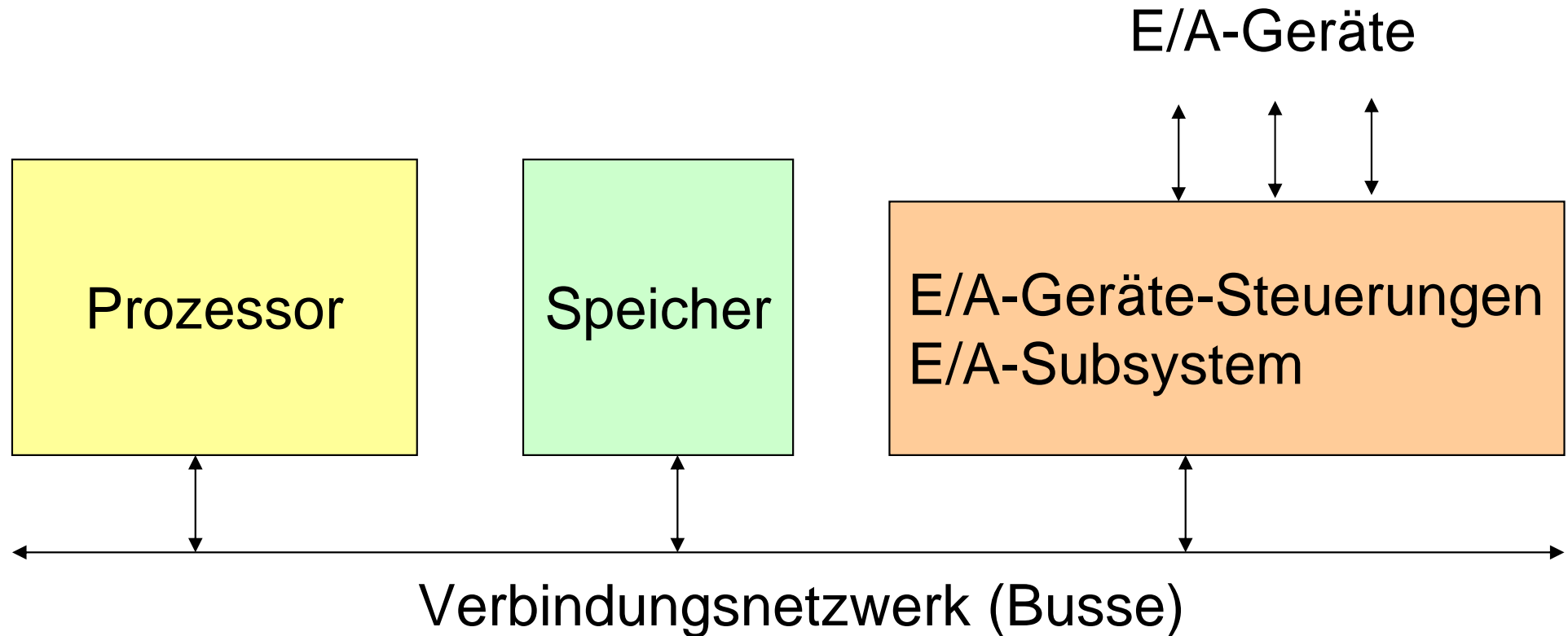
# Organisation von Speicherbausteinen

---

- Durch das Anlegen von Eingangssignalen wird eine Speicherzelle ausgewählt (adressiert) und der dort gespeicherte Funktionswert an den Ausgängen zur Verfügung gestellt.
- Die Leitungen, die den Dekoder verlassen, entsprechen den Mintermen von  $n$  Eingangsvariablen, also den Zeilen der Funktionstabelle.
- Das Speichern einer **1** für eine bestimmte Ausgangsvariable  $i$  bedeutet, dass dieser Minterm in die ODER-Verknüpfung am  $i$ -ten Ausgang einbezogen wird, eine **0** heißt, dass der Minterm nicht benutzt wird.

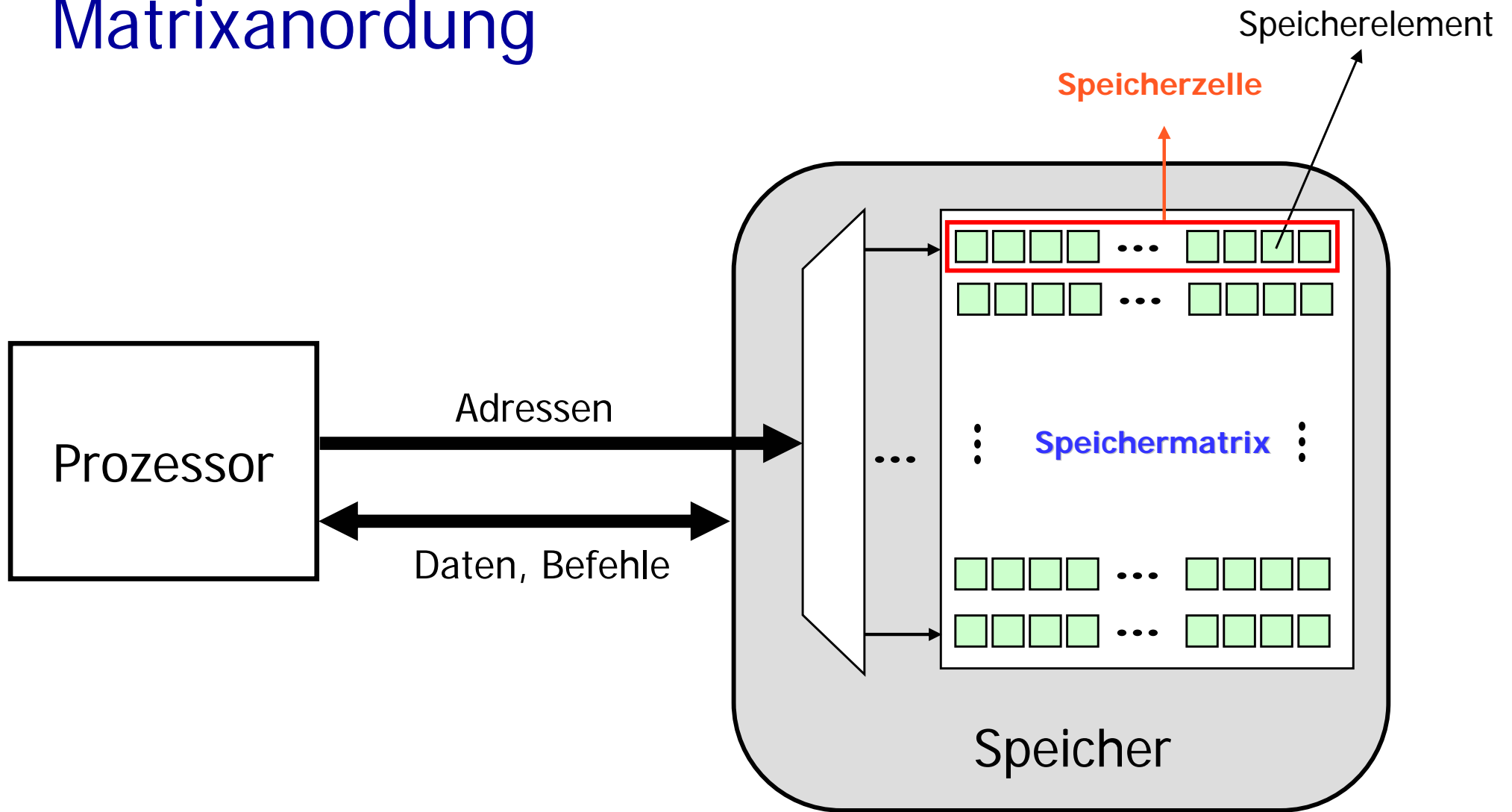


# Speicherwerk im Rechner

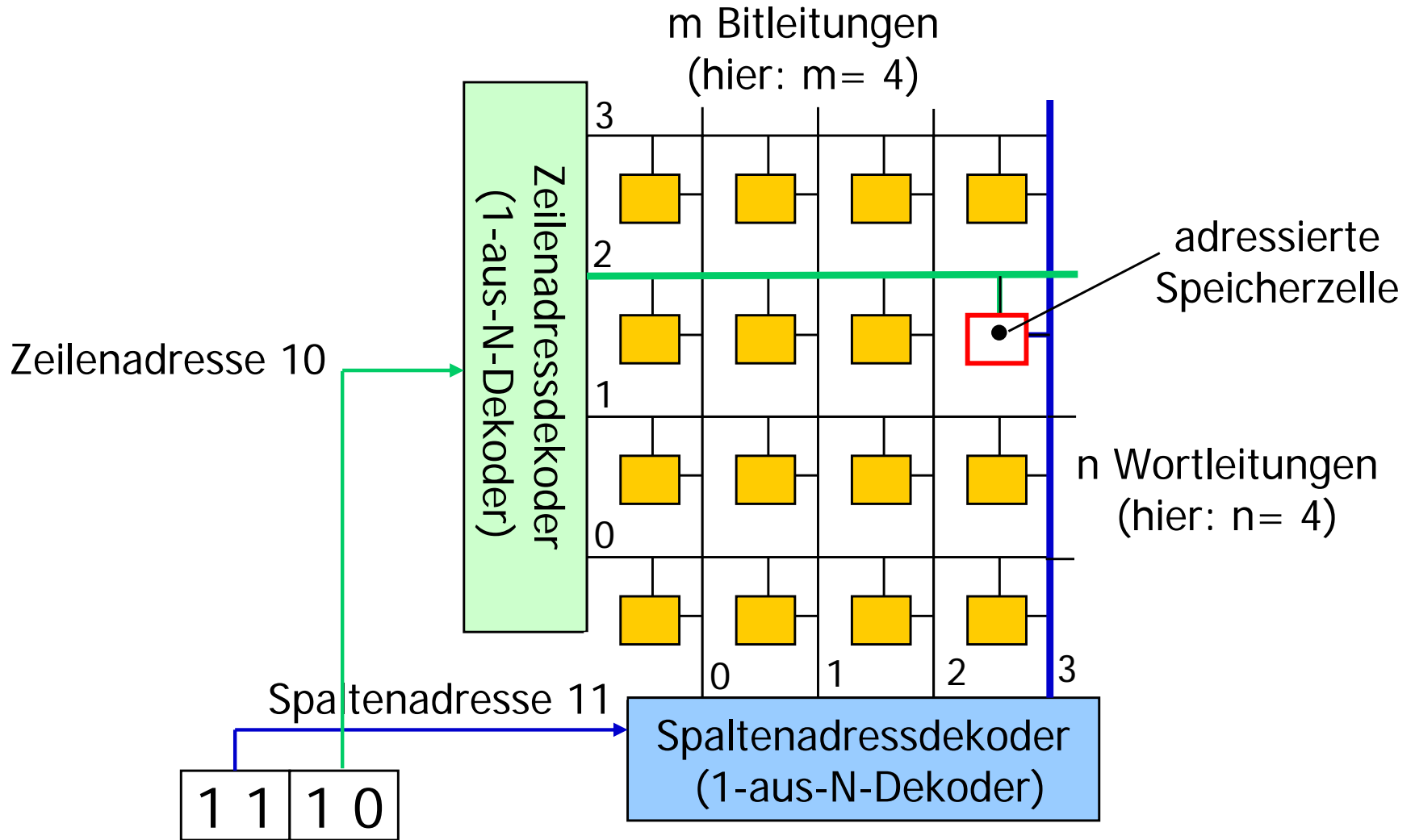


# Speicherwerk: Allgemeine Struktur

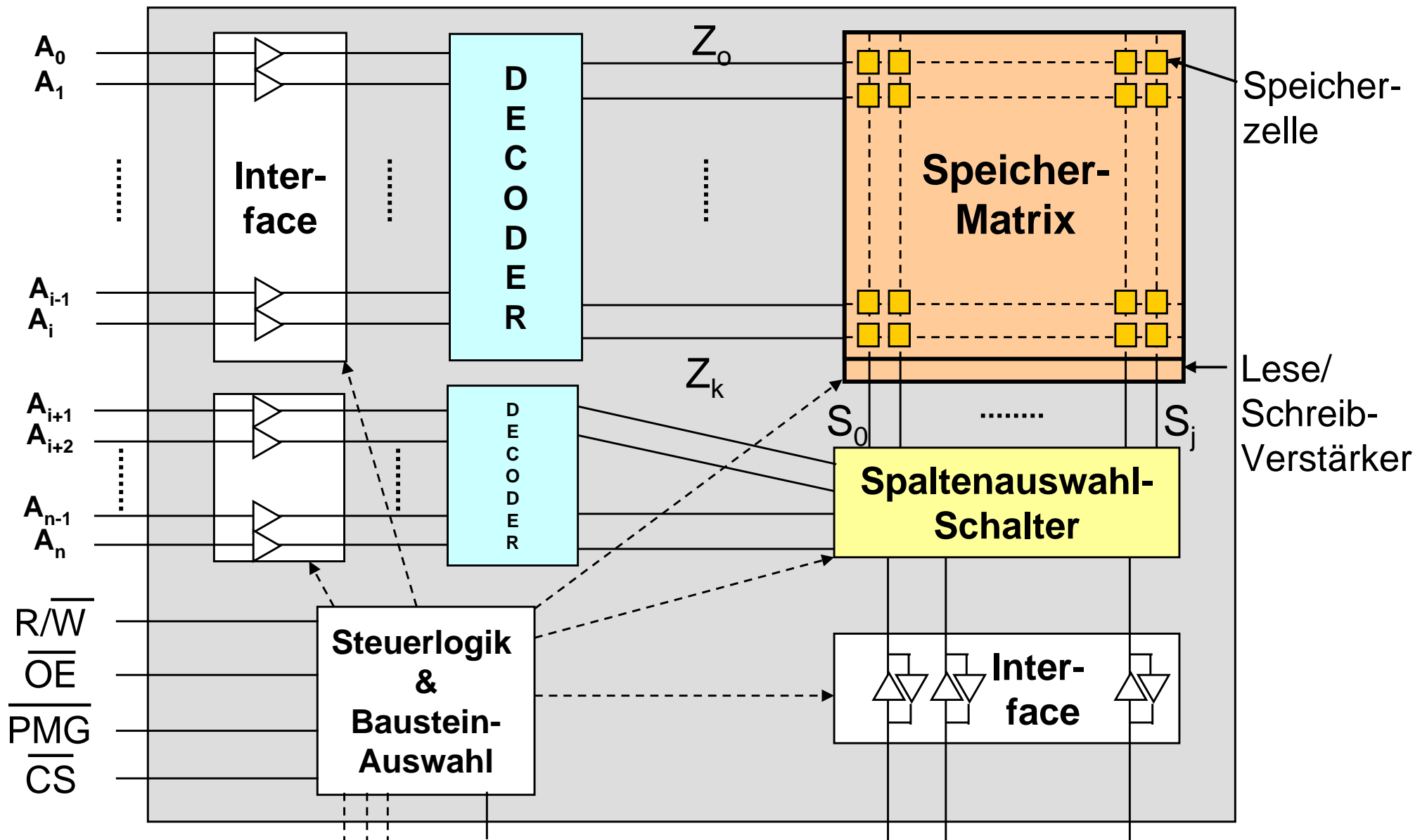
## Matrixanordnung



# Beispiel: Selektieren einer Speicherzelle aufgrund der gegebenen Speicheradresse



# Organisation von Speicherbausteinen



# Beispiel

Ein Schaltnetz soll für alle 3-Bit-Eingangszahlen ihre Quadrate am Ausgang liefern.

**Eingänge:**  $e_2, e_1, e_0$

**Ausgänge:** max. 6 Ergebnisstellen  $a_5, a_4, a_3, a_2, a_1, a_0$

**Funktionsstabelle :**

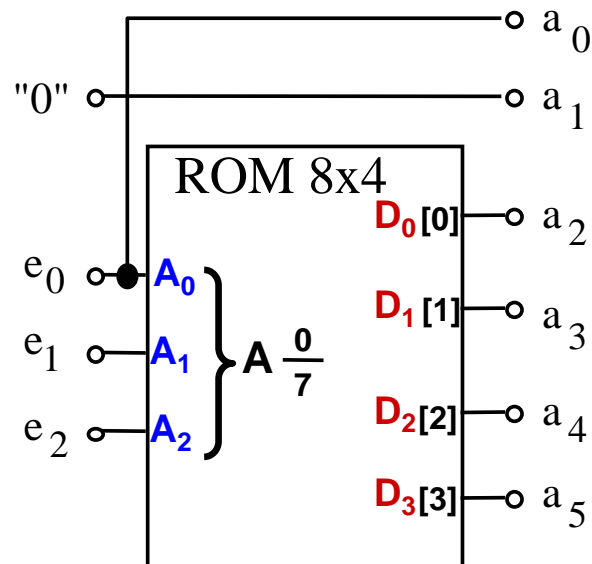
i	$e_2$	$e_1$	$e_0$	$i^2$	$a_5$	$a_4$	$a_3$	$a_2$	$a_1$	$a_0$
0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	1
2	0	1	0	4	0	0	0	1	0	0
3	0	1	1	9	0	0	1	0	0	1
4	1	0	0	16	0	1	0	0	0	0
5	1	0	1	25	0	1	1	0	0	1
6	1	1	0	36	1	0	0	1	0	0
7	1	1	1	49	1	1	0	0	0	1

# Lösung

$a_1$  und  $a_0$  brauchen nicht mit dem Speicherbaustein erzeugt zu werden, sie können direkt abgelesen werden:

$$a_1 = 0, a_0 = e_0$$

➔ Zwei Ausgänge beim Speicherbaustein werden einspart.

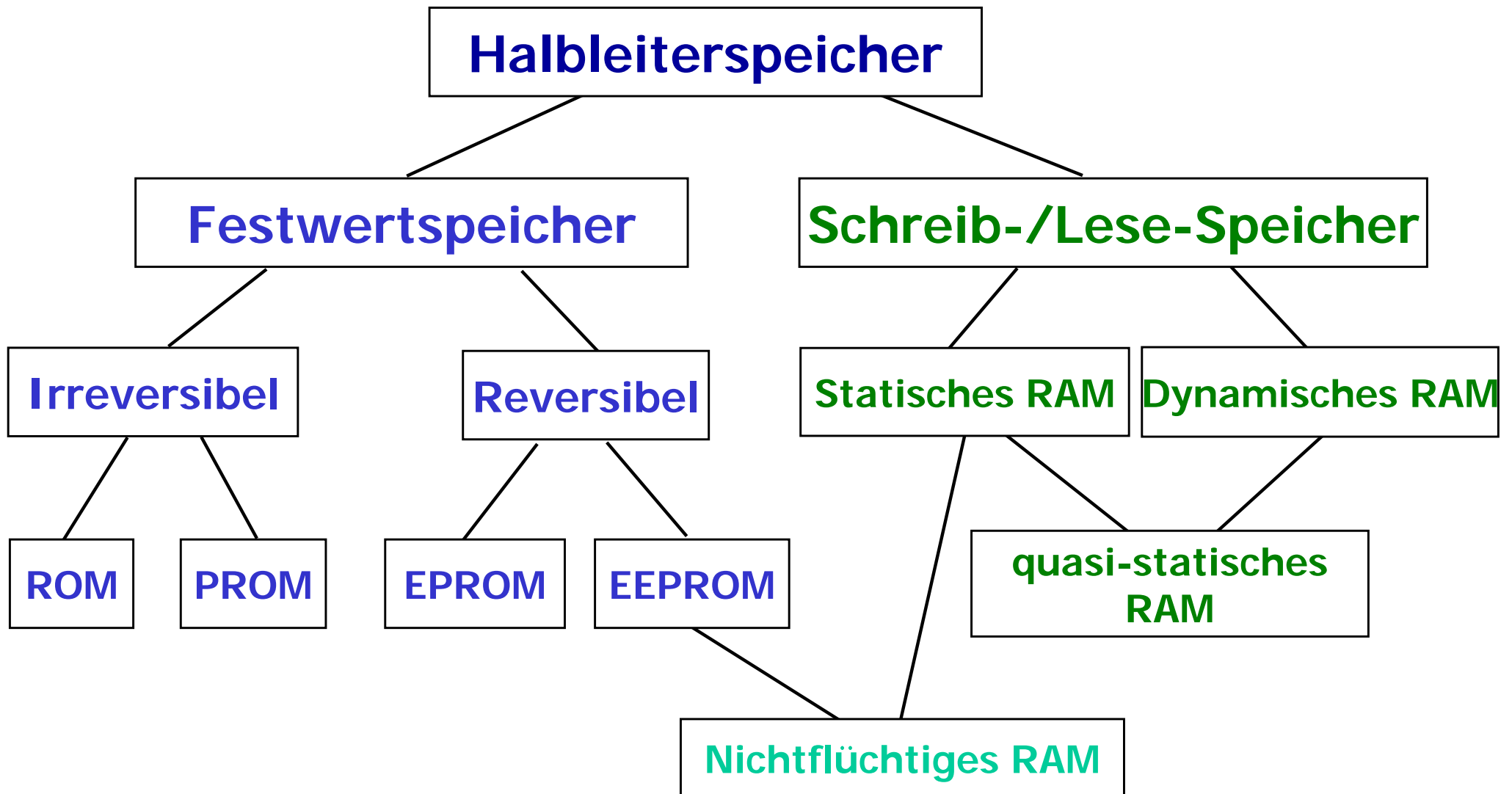


$A_2$	$A_1$	$A_0$	$D_3$	$D_2$	$D_1$	$D_0$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

Die Speicherbelegung entspricht (bis auf die 2 weggelassenen Spalten) genau der Funktionstabelle.



# Speichertypen



# Festwertspeicher (ROM, *Read Only Memory*)

---

Inhalt ist während des Normalbetriebs nur lesbar. Inhalt ist nicht flüchtig (*non volatile*), d. h. er geht bei Abschaltung der Versorgungsspannung nicht verloren

## □ **Irreversible Festwertspeicher:**

Das Einschreiben einer Information kann nicht wieder rückgängig gemacht werden

- maskenprogrammierten Festwertspeicher (ROM):  
Programmierung bei der Herstellung, nur bei großer Stückzahl
- programmierbare Festwertspeicher (PROM):  
einmaliges Programmieren durch den Anwender mit Programmiergerät (Durchbrennen von Verbindungen, fusable links)



# Festwertspeicher (ROM, *Read Only Memory*)

---

## □ **Reversible Festwertspeicher:**

Das Einschreiben der Informationen kann wieder rückgängig gemacht werden.

- UV-löschbaren Festwertspeicher (EPROM):  
Löschen durch UV-Licht.
- Elektrisch löschbare Festwertspeicher (EEPROM):  
können im  $\mu$ P-System selbst elektrisch gelöscht werden.  
Dieser Vorgang ist jedoch sehr langsam und nur begrenzt oft möglich.
- Flash-ROM:  
Werden genauso wie EEPROMs elektrisch programmiert. Die Daten werden nicht byteweise sondern blockweise ausgelesen und geschrieben.



# Schreib/Lese-Speicher (RAM, *Random Access Memory*)

---

- ❑ Random Access Memory (RAM): Zugriffszeit unabhängig vom Ort (Adresse) im Speicher
- ❑ Inhalt ist jeder Zeit lesbar und schreibbar.
- ❑ Inhalt ist flüchtig (*volatile*), d. h. er geht bei Abschalten der Versorgungsspannung verloren.
  - **Statische Schreib/Lese-Speicher (SRAM)**
  - **Dynamische Schreib/Lese-Speicher (DRAM)**



# Schreib/Lese-Speicher (RAM, *Random Access Memory*)

---

## **RAM (Random Access Memory):**

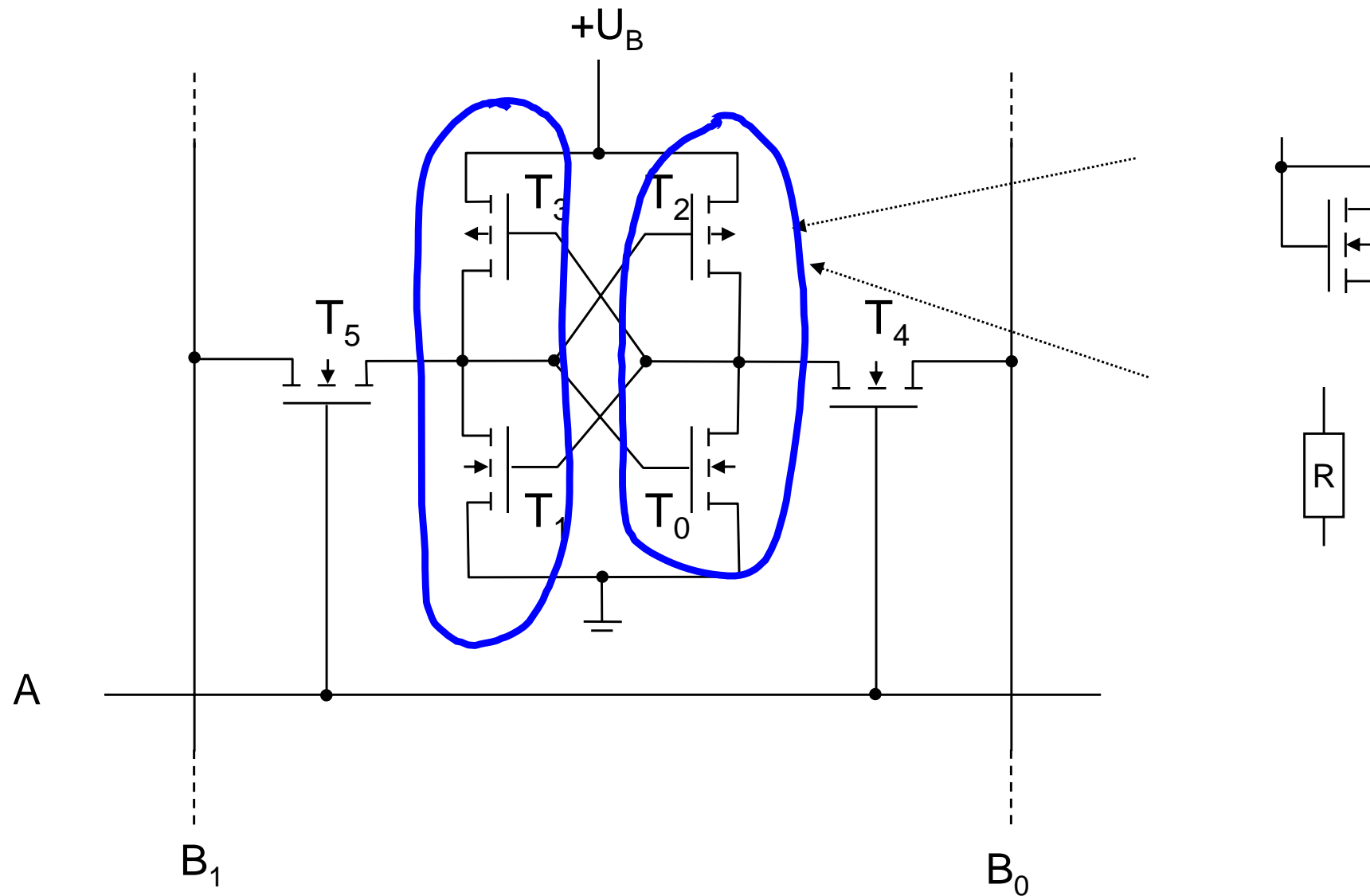
Speicher, auf die während des Normalbetriebs lesend und schreibend zugegriffen werden kann.

Man unterscheidet :

- **Statische RAM-Bausteine (SRAM)**
- **Dynamische RAM-Bausteine (DRAM)**

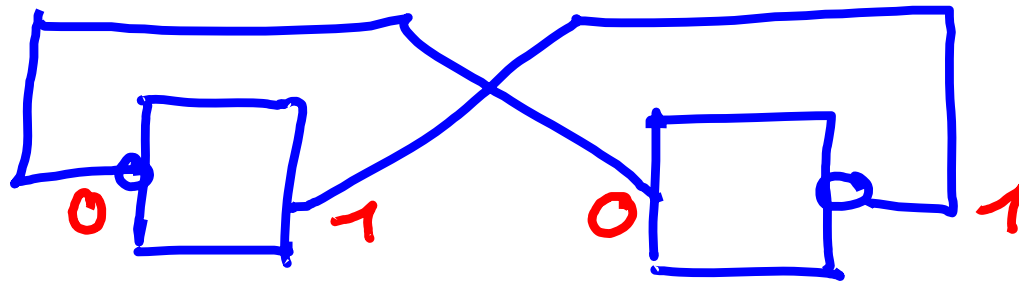
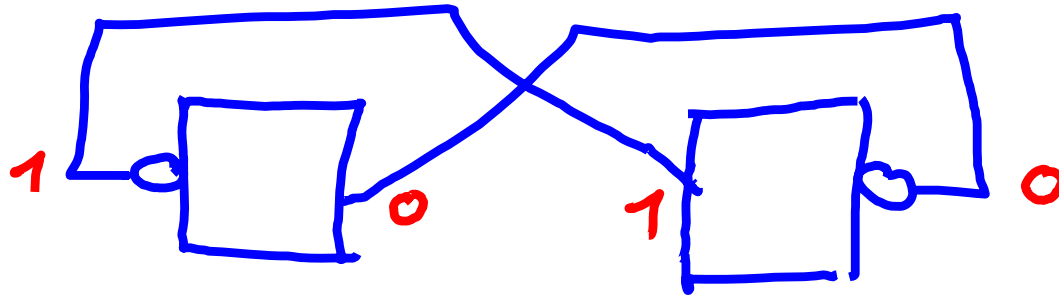


# Statische RAM-Speicherzellen (SRAM)



# Ergänzung

Kreuzweise rückgekoppelte Inverter



# Statische RAM-Speicherzellen (SRAM)

---

Eine Statische CMOS-Speicherzelle besteht aus

- 2 kreuzweise rückgekoppelten Invertern ( $T_0, T_2$  bzw.  $T_1, T_3$ ).
- 2 Transistoren  $T_4$  und  $T_5$  zur Ankopplung an die Bitleitungen

➔ **6 Transistor Zelle**

## Vorteil der CMOS-Zelle:

nur zum Umschaltzeitpunkt fließt Strom

NMOS-Zelle:

Inverter aus n-Kanal-Transistor und Widerstand  
wird benutzt (Varianten b, c)





# Dynamische RAM-Speicherzellen

---

## Lesen:

**Problem:** Speicherkapazität hat ungefähr die gleiche Größe wie parasitäre Leitungskapazität der Bitleitung

Das Lesen erfolgt durch Ladungsvergleich zwischen Leitungskapazität und Speicherkapazität

## Ablauf:

- Zunächst wird die Leitungskapazität vorgeladen (*precharge*), indem die Bitleitung kurz über  $T_L$  mit  $+U_B$  verbunden wird
- Zum Lesen wird dann über A eine positive Spannung an das Gate des Speichertransistors gelegt

Ist die Speicherkapazität geladen, so findet ein Ausgleich mit den Ladungsträgern der Bitleitung statt

Leseverstärker am Ende der Bitleitung mißt diesen Strom



# Dynamische RAM-Speicherzellen

---

## Schreiben:

Durch Anlegen einer positiven Spannung  $U_{GS}$  wird der Speichertransistor leitend.

Liegt nun die Bitleitung B auf Masse

➔ Elektronen werden auf die Drain-Zone aufgebracht, der Speicherkondensator geladen

Liegt die Bitleitung B auf  $U_B$

➔ Elektronen werden von der Drain-Zone abgesaugt, der Speicherkondensator entladen

*(Zuordnung log. 0 / log. 1 zu Ladung / keine Ladung ist rein willkürlich)*



# Dynamische RAM-Speicherzellen

---

- Speichern Daten als elektrische Ladung in einem Kondensator
- Das Lesen bewirkt eine Entladung (*destructive read*)  
→ nach dem Lesen muss wieder eingeschrieben werden
- Die Ladung geht nach einiger Zeit auch durch Leckströme verloren → periodische Auffrischung erforderlich (*refresh*)
- Integrationsdichte höher als SRAM (ca. 4 mal)



# Vergleich SRAM und DRAM

---

- Als Speicherzelle wird bei SRAMs ein **Flipflop** verwendet.
  - ➔ Die Speicherzelle hält ihre Programmierung auch ohne Regeneration.
- Die Zugriffszeit bei einem statischen RAM ist wesentlich kürzer als bei einem dynamischen RAM.
- Eine statische Speicherzelle benötigt eine etwa 6 bis 8 Transistoren, eine dynamische dagegen weniger Fläche.

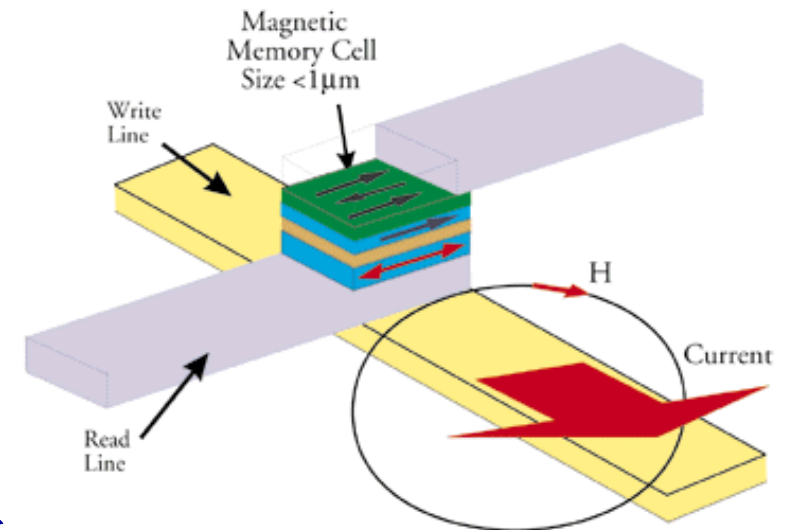


# Neue Entwicklung: MRAM

IBM und Infineon mit neuer Speichertechnik: **MRAM** (*Magnetoresistive Random Access Memory*) soll die Vorteile der verschiedenen RAM-Technologien vereinen:

- hohe Geschwindigkeit von SRAMs,
- Dichte und die Kostenvorteile von DRAMs,
- nichtflüchtige Speicherung von Flash-Speichern
- Magnetische statt elektronischer Ladungselemente für die Speicherung
- Mit der MRAMs kann z. B. der Bootvorgang bei PCs und Mobilgeräten entfallen.

*VLSI Symposium in Kyoto, Japan, Juni 2003*



# PLA (Programmable Logic Array)

---

Bisher wurde die gesamte Funktionstabelle in einem Speicherbausteinen abgespeichert und die Funktion durch ihre DNF realisiert.

Verwendet man stattdessen die DMF, lassen sich Funktionen oft sehr viel kompakter darstellen.

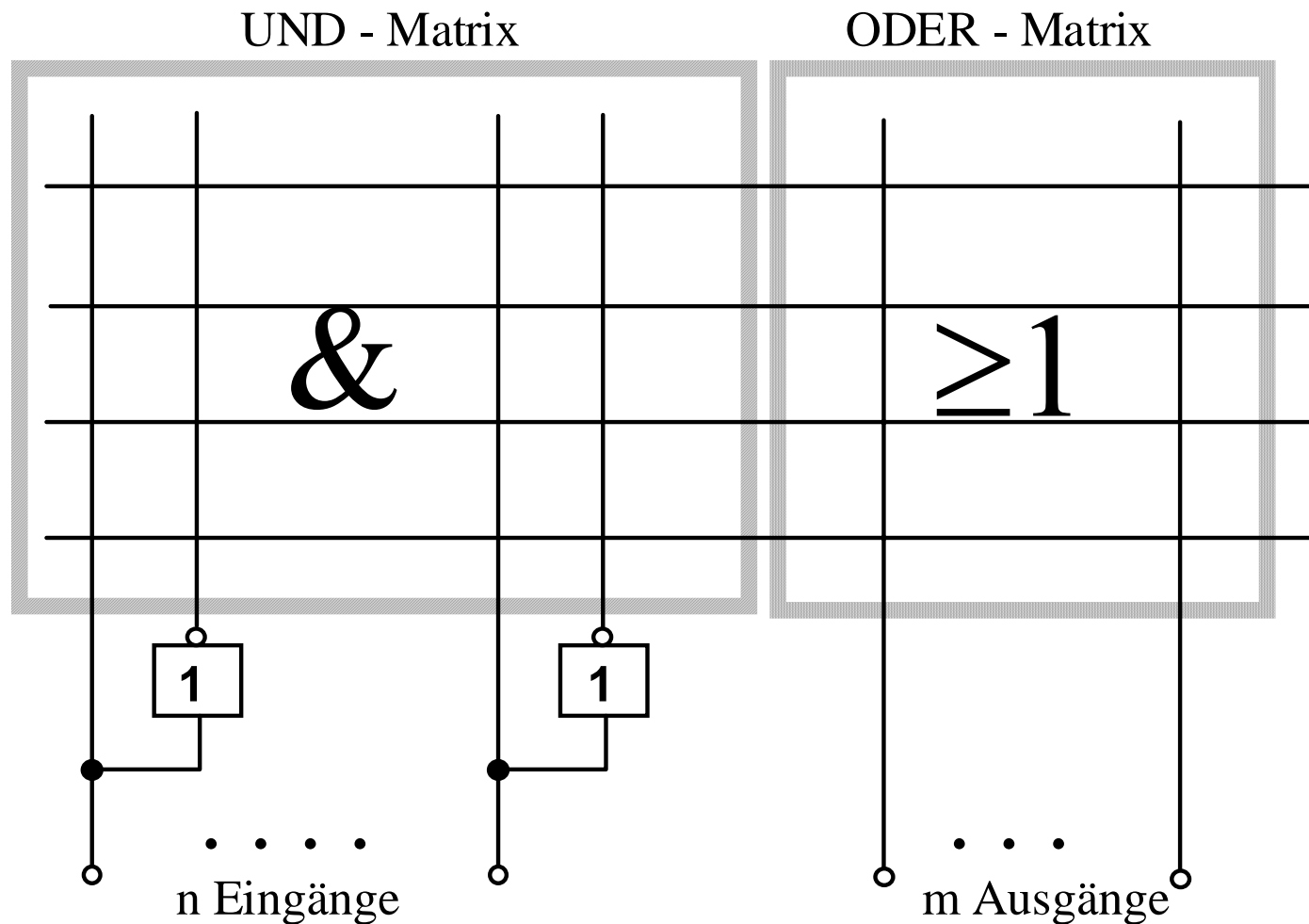
## **PLA (Programmable Logic Array):**

Im Unterschied zum ROM werden bei PLA eingangsseitig **nicht Minterme, sondern Primimplikanten** der Minimalüberdeckung erzeugt.

Dazu wird der Dekoder durch eine UND-Matrix ersetzt.



# Schematische Darstellung eines PLA

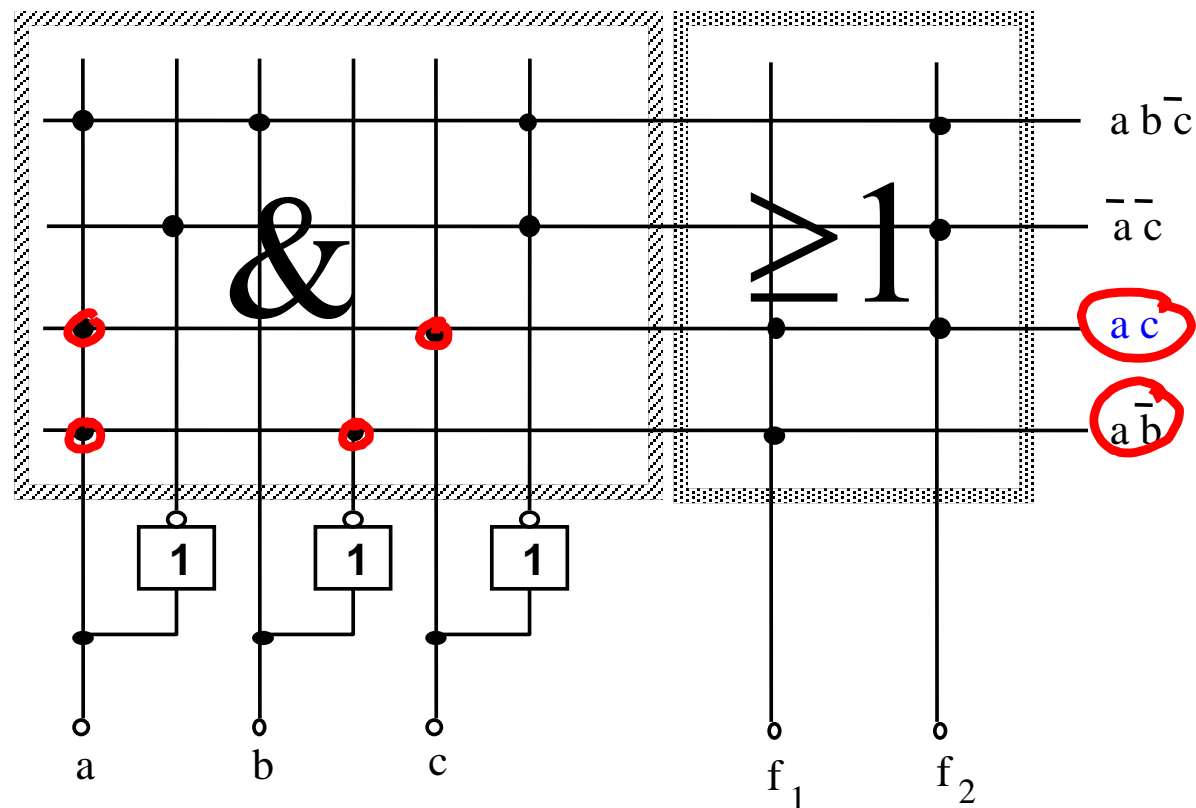


# Beispiel

Die (schon minimierten) Funktionen

$$f_1 = a \bar{b} \vee a c \quad \text{und} \quad f_2 = \bar{a} \bar{c} \vee a c \vee a b \bar{c}$$

sollen mit einem PLA realisiert werden.



# Bündelminimierung

---

Der Term  $a \cdot c$  im letzten Beispiel konnte für beide Funktionen verwendet werden, wodurch sich die Anzahl der notwendigen Produktterme reduziert.

Allgemein sollte beim Minimieren mehrerer Funktionen (Funktionsbündeln) immer darauf geachtet werden, nicht jede Funktion für sich zu minimieren, sondern den Gesamtaufwand für alle Funktionen zu optimieren.  
(Bündelminimierung)



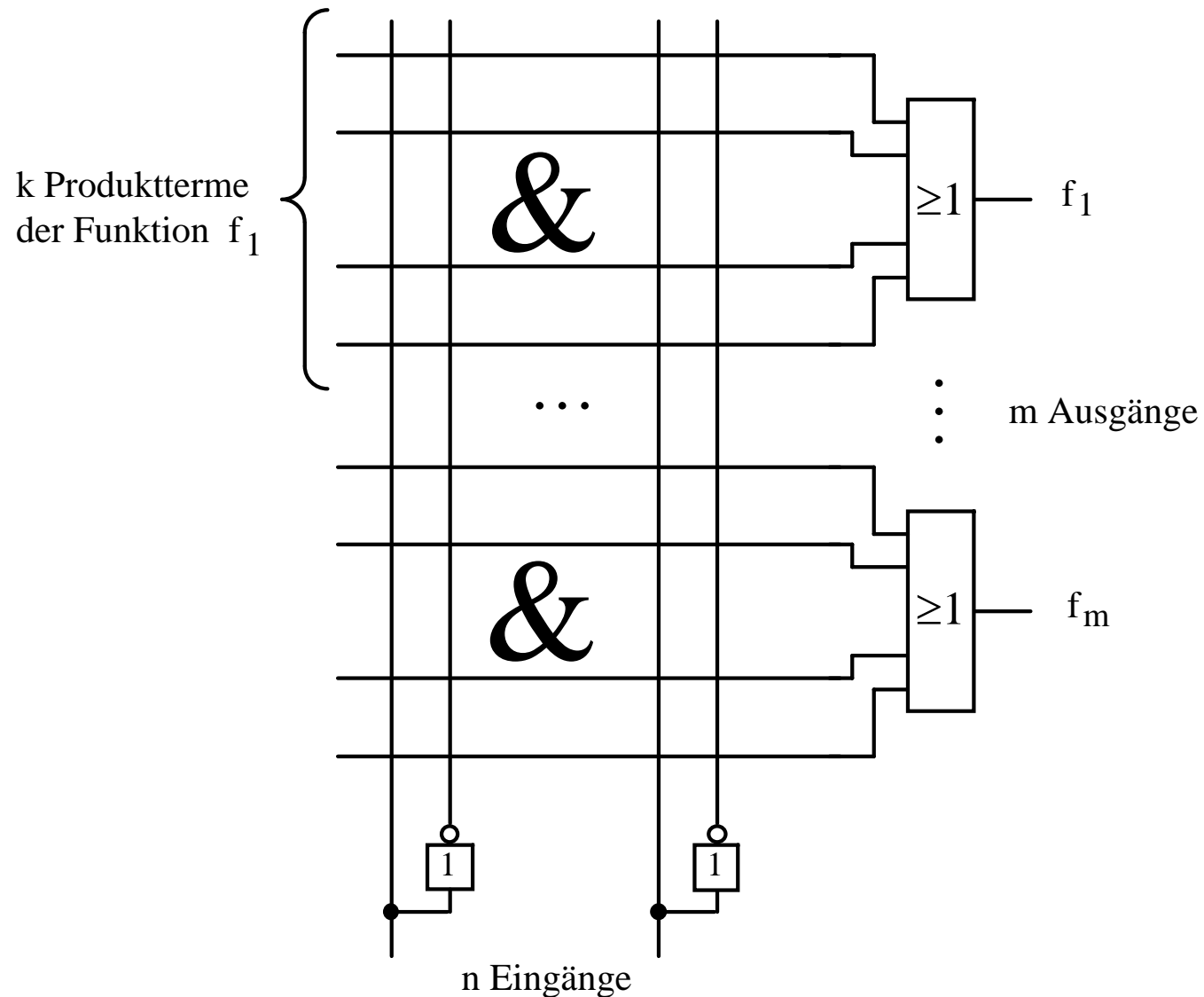
# FPLA und PAL

---

- PLAs werden ähnlich wie ROMs bereits bei der Herstellung personalisiert.
- Ein vom Benutzer zu programmierendes PLA mit fest vorgegebener Anzahl von Eingangsvariablen  $n$ , Produkttermen  $k$  und Ausgangsvariablen  $m$  wird **FPLA (field programmable logic array)** genannt.
- Alternativ dazu werden **PAL-Bausteine** (*programmable array logic* oder *programmable and logic*) angeboten, bei denen die ODER-Matrix bereits in der Herstellung personalisiert wurde.

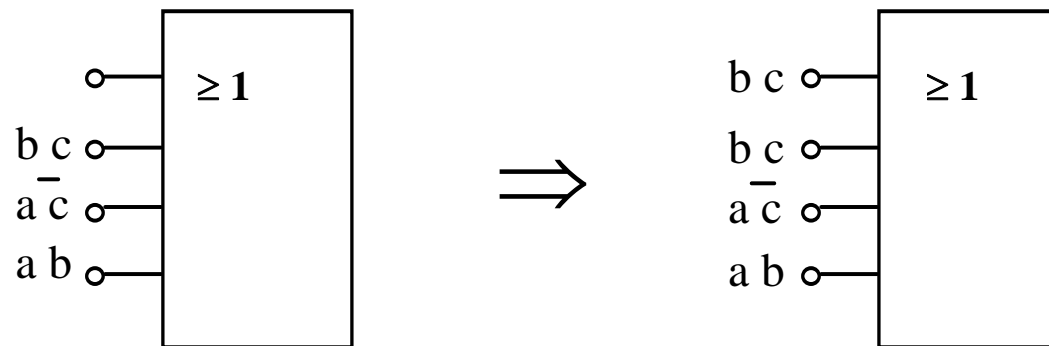


# Schematische Darstellung eines PAL



- Verschiedene **PAL-Typen** unterscheiden sich dabei sowohl in der Anzahl der Eingänge und Produktterme, als auch in der Programmierung der ODER-Matrix.
- Die Anzahl der Eingänge der ODER-Gatter von PALs ist damit fest vorgegeben.
- Unbenutzte Eingänge müssen deshalb mit einem "Dummy-Term" belegt werden, der die Funktion nicht verändert.

### Beispiel:





Schöne Feiertage  
und ein erfolgreiches Jahr 2006  
wünscht Euch das TI-Team

